sas

循序渐进掌握数据分析要义和精髓
从数据中获取洞见与智慧

SAS 技术内幕

从程序员到数据科学家

巫银良◎著

INSIDER OF SAS TECHNOLOGY

FROM PROGRAMMER TO DATA SCIENTIST

张铭
北京大学信息科学技术学院
博士生导师、教授

刘云中
国务院发展研究中心
研究员

唐朝晖
艾德思奇（adSage）
董事长

刘政
SAS中国研发中心
总经理

李涛
慧科集团技术产品中心
技术副总裁

谷鸿秋
《SAS 编程演义》
作者

付春光
北京冲和资产管理有限公司
量化投资总监

杨祉雄
致远互联
高级副总裁

林建伟
华为供应链管理部
数据科学家

陈云凯
SAS 中国销售总监兼合作伙伴
与渠道负责人

联袂诚意推荐

清华大学出版社

SAS 技术内幕：从程序员到数据科学家

巫银良　著

清華大學出版社
北　京

内容简介

本书共27章，分为上下两卷：上卷介绍SAS编程基础与使用方法，是广大程序员快速掌握SAS编程技术的简明开发教程；下卷阐述数据分析的关键基础知识并提供相应SAS代码实现，目的是激发读者兴趣，助其跨越传统编程与数据分析的鸿沟，从程序员华丽转身为数据科学家。书中演示代码力图简洁清晰地解释相关概念，追求大道至简。本书兼顾编程技术与数据分析，期待程序员、信息处理与统计分析人员以及对数据分析科学感兴趣的读者都能从本书中获益良多，循序渐进地掌握数据分析的要义和精髓，从数据中获取洞见与智慧。

图书在版编目 (CIP) 数据

SAS技术内幕：从程序员到数据科学家 / 巫银良著. — 北京：清华大学出版社，2018
ISBN 978-7-302-50278-4

Ⅰ. ①S… Ⅱ. ①巫… Ⅲ. ①统计分析—应用软件—程序设计 Ⅳ. ①C819

中国版本图书馆CIP数据核字（2018）第111953号

责任编辑：刘　洋
封面设计：李召霞
版式设计：方加青
责任校对：宋玉莲
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦A座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：
经　　销：全国新华书店
开　　本：185mm×260mm　　印　　张：38.75　　字　　数：891千字
版　　次：2018年12月第1版　　印　　次：2018年12月第1次印刷
定　　价：168.00元

产品编号：077684-01

All knowledge is, in final analysis, history.
在终极的分析中，一切知识都是历史

All sciences are, in the abstract, mathematics.
在抽象的意义下，一切科学都是数学

All judgements are, in their rationale, statistics.
在理性的基础上，所有判断都是统计

—— Calyampudi Radhakrishna Rao ——

All models are wrong, but some are useful.

模型皆有误，或尤建奇功

——George E. P. Box——

谨以此书献给远在家乡的母亲，
她劬劳一生且广受尊敬……

推荐语

《SAS 技术内幕：从程序员到数据科学家》是值得一读的数据分析技术入门佳作。这本书是数据分析领域资深专家为融合程序员和统计分析人员的不同视角而编写的，给读者在计算机程序设计世界和数据分析世界之间架起一座桥梁……罗马不是一天建成的！但阅读本书将使普通的程序员也能快速入门 SAS 语言并以最简洁的方式掌握 SAS 编程核心，实现向优秀数据科学家的华丽转身！

北京大学信息科学技术学院博士生导师、教授　张铭

本书把复杂的数据分析科学以最简单的应用和实例介绍给读者，是一本难得的告诉你用 SAS 将数据变为知识，将知识变为决策的好书！

国务院发展研究中心研究员　刘云中

SAS 是数据分析、数据挖掘和人工智能的领军平台。本书集结了作者多年从事数据分析和商业智能研究与实践的经验。用 SAS 平台深入浅出地给程序员介绍了数据挖掘的基础与精髓，帮助他们成为数据分析的真正专家，非常推荐！

艾德思奇（adSage）董事长　唐朝晖

现在很多企业都开始进入数据分析领域，由于没有现成的、足够多的数据分析人员，企业只能让现有的软件开发人员承担这些责任。然而，软件开发人员与数据分析人员的思维是有差异的，也就是确定性的结果和概率性的结果的差异。如何帮助这些软件开发人员成为合格的双重人才，需要一定的知识补充和理论指导。巫银良先生根据自己丰富的从业实践与行业观察总结出来的这本书正好填补了这一空白，值得推荐！

SAS 中国研发中心总经理　刘政

SAS 作为一门古老而高效的统计编程语言，在商业数据分析行业已经流行了几十年，历久弥新！它给人的感觉是神秘而复杂，深奥且与众不同，往往让人难以理解！但本书作者在多年编程经验和分析实践的基础上，用简单直白的语言将数据分析这件事说明白了。本书从程序员的角度出发，以简洁实例带领读者从程序员世界深入数据分析的世界，是一本写给程序员的数据分析技术入门佳作！

慧科集团技术产品中心技术副总裁　李嘉

很多SAS统计师其实是野蛮生长起来的，比如我，面对代码缺乏程序员的严谨思维；很多SAS程序员其实是半道出家的，比如我，面对数据缺乏统计师的分析思维。巫银良先生此书恰好弥合了这两方面的话题。本书不仅对常规SAS编程语言进行了细致的介绍，而且对各种数据结构用SAS编程语言进行了对接；不仅有常规统计方法的讲解，而且有统计方法背景知识的SAS论证，末章更是趣味大增、思路大开，花式展示用SAS高精度求解 π 值。本书内容丰富，构思严谨，必将是程序员、统计分析师以及未来的数据科学家们手中的一份上好读物。

《SAS编程演义》作者　谷鸿秋

本书由基础到专业、循序渐进地介绍了数据分析的各种技术和应用，是从事量化投资专业人士必读的入门书籍。

北京冲和资产管理有限公司量化投资总监　付春光

企业数字化时代已到来！当每个企业每个人每个物体都在被不断数据化时，对数据化背后的洞察获取相比高速增长的数据积累显得迟钝。这本书将弥补这一不足，让更多程序员、统计分析师、软件产品设计者向数据科学专业人士方向发展，以科学的方法去发现数据的神秘和数据之美。

致远互联高级副总裁　杨祉雄

当我看到标题的一刻，心想这可能又是一个专门针对程序员讲解编程的书。但当我打开书，慢慢品读的时候，有的是惊喜；在书中，可以看到与数学概念相关的历史、公式，计算机技术以及SAS编程的有机融合。我一口气读下来，没有感到晦涩和枯燥，这是一本值得推荐的好书。

华为供应链管理部数据科学家　林建伟

SAS的合作伙伴，期待一些系统化地解释SAS技术和应用案例的书籍，以帮助合作伙伴提升SAS应用技能，从而更好地服务客户，实现商业价值。本书有层次、有节骤地帮助SAS顾问和技术人员，兼顾程序员视角和数据科学家视角，逐步深入地探索SAS技术的应用，以充分发挥SAS技术在商业分析具体业务场景中的优势，这是一本非常值得推荐的好书！

SAS中国销售总监兼合作伙伴与渠道负责人　陈云凯

序　言

《SAS 技术内幕：从程序员到数据科学家》是值得一读的数据分析技术入门佳作。这本书是数据分析领域资深专家为融合程序员和统计分析人员不同视角而编写，给读者在计算机程序设计世界和数据分析世界之间架起一座桥梁。

我在北京大学计算机系从事计算机教学和科研几十年，所主讲的“数据结构与算法”被评选为国家级精品课程。图灵奖获得者尼古拉斯 • 沃斯 (Niklaus Wirth) 提出“程序 = 算法 + 数据结构”，而计算机语言则是构建程序世界的主要工具。在计算机编程语言世界里，除了汇编、C/C++ 和 Java/C# 等主流通用语言外，还有很多面向特定领域的高级语言，如 SAS、R 和 MATLAB 等。我一直鼓励学生们开阔视野，多接触一些工业界的专业工具和编程语言，不要局限在 C/C++ 等通用语言的框架里。在这些专用语言的背后，往往蕴含特定的领域思维和设计哲学，基于专业人士在处理领域问题时积累的丰富经验进行了极其灵活的设计，专用语言的这些特性往往是通用编程语言所不考虑也不具备的。

近些年来，数据分析和商业智能发展迅猛，大数据和人工智能在学术界和产业界都生机勃勃。学术界科研人员注重的是引领世界科技发展的超前研究，在理论和创新方面有独到之处。工业界是前沿技术的成熟应用，学术界在培养应用型人才方面需要考虑工业界的真实需求。因此，北京大学很早就与全球数据分析行业的领导者 SAS 合作，开设了面向研究生和高年级本科生的统计分析选修课程“统计分析与商务智能”，取得了非常好的教学效果。SAS 是全球数据分析领域的领导者，它们以创新的软件和服务，在数据分析、商业智能、数据管理等领域耕耘四十余年，一直秉持的理念就是提供“慧识力量”（The Power to Know®），使用户能够对海量数据深入了解并获得洞见和价值（Insight & Value），为企业的运营发展提供决策利器。

如果按照数据的生产和消费进行划分，大部分应用软件和系统，包括互联网社交媒体和电子商务都在大量制造数据，这一进程经过信息化时代和互联网时代已经得到充分体现。如何消费这些数据却催生了其他软件系统的发展，它们包括各种分析系统如商业智能（BI）、决策系统、专家系统和人工智能系统（AI）等。近些年来数据分析在公众媒体眼中变得非常热门，原因就是分析利用数据并从中获取价值在“信息化时代”和“互联网时代”之后的“大数据时代”变得尤为迫切。在过去的几年中，处理数据、对数据进行可视化分析、理解数据，跟数据进行沟通、深入探索，并从中获得价值，这些过程已经变得极为重要。**在软件领域，数据分析将是 21 世纪头十年经过商业智能产业大并购之后的下一个蓝海！不过，蓝海中不再是程序员划着小船在徜徉，而是数据科学家们开着利舰遨游在大数据的海洋之上。**

现实已经告诉我们，计算机程序员需要在程序员思维之外，尽快拥有数据分析思维，

华丽转身成为数据科学家！文本分析、语音识别、神经网络、人工智能、自动驾驶等各种最新最热门的数据分析领域，说到底需要的是强大的综合能力，包括良好的计算机编程技能（Programming）、扎实的数学和统计知识（Mathematics & Statistics）、专业的业务领域知识（Business）以及与数据来源和消费方良好的沟通技能（Communication）等。除具备良好的编程能力和丰富的数据库知识之外，如果程序员掌握像 SAS 这样一种严谨且专门面向数据分析的语言，拥有扎扎实实修炼数学和统计分析的能力，那就能让自己站在数据分析的最前沿。

罗马不是一天建成的！但阅读本书将使普通的程序员也能快速入门 SAS 语言并以最简洁的方式掌握 SAS 编程核心，实现向优秀数据科学家的华丽转身！

北京大学信息科学技术学院
博士生导师、教授
张　铭
2018 年 8 月

前　言

本书是写给程序员的数据分析技术入门书籍，成书于2017年作者在北京大学教授面向研究生和高年级本科生的“统计分析与商务智能”选修课期间，它试图在程序世界和数据分析世界之间架起一座坚实的桥梁。

本书主要包括上下两卷内容：程序员视角下的SAS编程技术和数据结构，数据科学家视角下的数据分析理论和SAS实践。

上卷主要包括SAS语言入门、数据集与DATA步、变量与表达式、流程控制、函数封装、SAS宏、DS2、代码组织、文件读写、按位运算以及扩展SAS功能。另外，还从程序员的视角阐述了各种数据结构在SAS中的编程实现和应用，包括SAS数组、队列与堆栈、链表、二叉树、矩阵运算和图等。

下卷包括统计学基础、大数定律与中心极限定理、统计分布、方差分析、数据标准化、主成分分析与因子分析、相关分析与回归分析、聚类分析、神经网络，最后以π值高精度求解和探索分析结束。

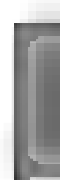
本书内容涵盖理论和实践，章节组织采用从简单到复杂的方式。本书囊括数据分析技术方面较为核心的基础内容，试图与读者一起触及数据世界分析与智能的核心。各章为读者提供简洁可运行的SAS示例代码、算法实现以及快速指南，为广大受过计算机科学教育的程序员向数据科学家华丽转身提供了必要的快速入门指导。本书附录还提供二项分布、泊松分布和标准正态分布累积概率表以及t分布、χ^2分布和F分布临界值表的制作与查找方法。本书适用于各大专院校统计分析专业和信息处理专业的学生，有志于从事数据分析的广大程序员、统计分析从业人员以及所有想成为数据科学家的专业人士。

本书与传统的SAS数据分析书籍不同之处在于，它从程序员的视角出发，循序渐进探讨数据分析的各个方面，避免“知其然而不知其所以然”。因此，阅读本书可使你从完全不了解SAS到对SAS“有所了解”，从而掌握数据分析的要义和精髓，实现从程序员到数据科学家的华丽转身！

最后，诚挚感谢北京大学信息科学技术学院博士生导师、中国ACM教育专委会主席张铭教授为本书作序。感谢SAS中国研发中心总经理刘政博士给予的大力帮助和指导，感谢SAS大中华区前市场总监蒋顺利先生、高级市场经理曾秋媚女士在写作过程中给予的帮助和支持，感谢在写作过程中给予特别帮助的SAS中国研发中心的同仁们。感谢清华大学出版社编辑部的刘洋先生和全体同仁的辛勤工作，是他们的努力使本书得以与读者见面！感谢恩师陈永金先生以及所有在我生命中给予帮助的人们！特别感谢与我一路同行的知己和家人，是他们赋予了我生命的色彩与力量！

巫银良

2018年8月



目　录

上　卷

下　卷

上　卷

第 1 章

SAS 语言入门

SAS 是英文 Statistical Analysis System 的简称，英文发音为 /sass/。它具有多层含义：首先，SAS 作为一家高科技常青藤软件公司，由参加过阿波罗计划的统计学家 James Goodnight 博士和 John Sall 博士成立于 1976 年，总部位于美国北卡罗来纳州的卡利市。James 从 1976 年 7 月 1 日起担任首席执行官至今，在 2004 年曾被哈佛大学提名为伟大的美国商业领袖。目前，SAS 是全球商业分析软件和服务的终极领导者，是商业智能和数据分析行业最大的独立供应商。其次，SAS 作为一个庞大的软件系统，以提供"慧识力量"（The Power to Know®）为己任，向全球各行各业提供全面的数据分析技术，涵盖统计分析、趋势预报、预测模型和优化等多个领域。在所有涉及数据分析的领域，SAS 公司都有非常强大而严谨的解决方案，是全球主要咨询服务商首屈一指的合作伙伴和分析基础设施提供者。SAS 自成立 40 多年来一直帮助用户和合作伙伴将各种数据转化为知识与智慧，从中发掘企业数据的商业价值。最后，SAS 作为一门历久弥新的计算机语言，自 1976 年起就是专门为数据处理和统计分析而设计的第四代计算机编程语言（4GL）。功能上主要面向数据科学的多个领域，包括数据操作、分析和报告。与传统的通用编程语言不同，SAS 是专门为数据科学设计的计算机语言，因此它具有自己独有的语言特征和运行模式。经过 40 多年的岁月洗礼，SAS 语言已经证明自己是数据分析领域的不二之选和事实上的标准。SAS 是美国联邦药品和食品管理局 FDA 接纳和审核电子提交材料以及财富 500 强生命科学公司的标准统计软件，以严格著称的 FDA 规定新药临床试验结果的统计分析只能用 SAS 进行，其他软件的计算结果一律不被承认。

SAS 的计算服务包括 SAS 语言、SAS 引擎和数据库服务三大核心模块。其中 SAS 语言是整个分析服务系统与用户进行交互的接口。

- SAS 语言：用于编写SAS 程序，主要由如下一系列面向数据操作和分析的语言元素构成。主要包括数据步（DATA Step）和过程步（PROC Step）。SAS步是现实世界中处理事情的步骤在程序世界里的反映。

（1）数据步（DATA Step）：负责数据读入、数据处理并创建SAS 能理解的数据表示，即数据集和数据列。

（2）过程步（PROC Step）：负责对数据进行分析，完成各种统计分析功能和图表报表功能。SAS 提供 400 多个功能强大的过程步，用于封装各种分析模块。

（3）SAS 宏（SAS Macro）：包括宏变量和宏函数，是实现 SAS 代码重用，减少

冗余代码逻辑的顶层利器。宏在很多计算机语言中都已经退化，但在SAS语言中依然保留了极其强大的生命力。

SAS语言是跨平台的编程语言，编写好的SAS程序可以运行在各种架构的Linux、UNIX、Windows甚至IBM大型主机上。SAS运行环境就像Java的JRE或.NET的CLR环境，它提供与主机平台无关的软件执行环境。实际上，SAS公司比SUN更早提出革命性的主机无关的可移植层概念，SAS代码跟Java代码一样"一次编写，随处运行"（Write once, Run anywhere），可跨平台运行在各种操作系统之上，SAS内部至今依然称这种架构为MVA（Multi-Vendor Architecture）架构。

与其他编程语言的I/O系统不同，SAS运行环境提供一个被称为输出交付系统（Output Delivery System，ODS）的部分，用于管理SAS程序的分析结果输出。ODS支持多种高级格式的输出目标（如LISTING、OUTPUT、DOCUMENT）以及多种第三方格式的输出目标（如HTML、PRINTER、MARKUP、RTF等），用户也可以自定义输出模板，来控制分析结果的形态和生成报告的类型，基于同样分析结果可以构建不同形式的报告。

- SAS引擎：为了让SAS语言编程人员专注于数据分析本身，SAS提供一系列引擎来屏蔽数据访问和执行环境相关的细节，让用户仅通过SAS逻辑库引用（Library Reference）就能标记需要分析的数据，而无须将太多的注意力放在具体的数据存储格式、数据库类型以及计算资源的形态等信息上。因此，SAS分析代码变得非常清晰、可重用，分析人员只需关心分析逻辑本身即可。

SAS引擎包括SAS逻辑库引擎（SAS Library Engines）和远程逻辑库服务（Remote Library Services）两大类，分别让用户在SAS代码中可以非常方便地访问本地磁盘上的数据或存储于远程计算机平台（如各种关系型数据库）的数据。根据SAS代码的运行模式，SAS传统上提供一系列运行方式不同的服务器，用于各种计算和分析：

（1）工作区服务器（Workspace Server）：SAS代码每次提交到该服务器运行，系统都会新建一个SAS运行环境，它对应操作系统的一个独立执行进程，通常称为SAS会话（SAS Session）。这跟SAS在单机环境上的运行模式大致相同：一个会话对应一个操作系统进程，每个会话具有独立的系统环境和临时数据区。

（2）存储过程服务器（Stored Process Server）：SAS代码多次提交到存储过程服务器运行，这些代码会共享一个SAS作为服务启动时建立的那个SAS会话，也就是多次代码提交共享单一会话模式，此时反复提交代码不会新建会话，而是重用已有会话。

（3）连接服务器（SAS/Connect Server）：可以让客户端机器充分利用服务端机器上的资源运行SAS程序并处理结果，构建"SAS到SAS"的客户端/服务端运行环境。

（4）网格服务器（Grid Server）：它是SAS应用服务器上用于桥接SAS应用和SAS网格环境的逻辑服务器，实现将繁重的计算作业分配到网格环境上并行执行。

- 数据库服务：包括数据访问、库内处理和内存分析几个层次，代表了数据访问模式的几个阶段。

（1）数据访问（Data Access）：对于各种各样的数据存在形态和数据库，SAS 都提供对应的数据访问服务来保证对特定数据的访问，SAS 支持从 PC 单机文件到关系型数据库、从分布式文件系统 HDFS 到云端（Cloud）数据的访问能力。

（2）库内处理（In-Database Processing）：SAS 为改进系统性能，减少数据在数据库管理系统和 SAS 运行环境之间的“传输和移动”，加速数据分析的开发部署而设计的一种高性能分析平台技术；它采用更加智能化的 SQL 来增强所选的分析过程步，在一些关键用例上甚至可将 SAS 系统函数直接部署到数据库运行环境的内部来执行。此时 SAS 嵌入进程与数据库本身的服务进程构成双头体系结构，共享同一数据。

（3）内存分析（SAS® In-Memory Analytics）：随着大数据时代的到来，高级分析和可视化分析呼唤对海量数据的大规模并发访问，海量并行处理（MPP）架构应运而生。SAS 运行在分布式计算环境的 LASR 分析服务器（LASR® Analytics Server）能够预先将海量数据加载到计算机网格的内存，提供安全、无状态的只读操作，从而实现在亚秒级完成对海量数据（亿行级）的分析。

2016 年，SAS 再次创新性地推出了 SAS 云分析服务（Cloud Analytics Services，CAS），它是 SAS 运行在云端最新的数据管理和分析运行环境，可直接对存储在云端（如亚马逊云、阿里云等）的商业数据进行分析处理。同时也是 SAS 在云平台开放时代的革命性创新，CAS 强大的分析服务支持不同的调用接口，用户只要使用 Java、Python 和 Lua 等语言 API 就可直接访问 CAS 服务，这在 SAS 历史上是前所未有的开放实践。

SAS 网格计算（SAS® Grid Computing）、库内计算（SAS® In-Database）和内存计算（SAS® In-Memory Analytics）是 SAS 高性能分析平台的三大支柱。近 40 年来，不管计算环境如何变化，SAS 一直致力于将最好的分析技术带给用户，为客户提供从数据（Data）到信息（Information），从知识（Knowledge）到智能（Wishdom）不断演进的“慧识力量”（The Power to Know®）。

数据分析的核心目的就是提升认知的过程。美国系统理论家罗素·艾可夫（Russell L. Ackoff）认为人的心智内容可分为数据、信息、知识、理解及智慧 5 个层次（图 1-1）。其中数据就是承载信息的原始符号，自身除了存在性以外没有任何意义。而信息就是数据通过关系连接在一起具有特定含义的有用数据，能够提供时间（When）、地点（Where）、人物（Who）以及什么（What）之类问题的答案。知识则是按照特定模式组合的信息集合，可以为人们所确定性地记忆和保存，能够回答事情是如何（How）发生之类的问题；然而知识还需要进一步建立在已知（历史）知识基础上对内插和概率性的理解，即从已知知识中获取新知识并综合新知识来回答为何（Why）的过程，才能达到认知和分析性的水准。知识和理解这两个层次就是“记忆”和“明白”的区别，就如小学生会背乘法口诀，但不能够处理两个较大数据的乘法运算一样。智慧则是一个不确定的非概率性外推过程，是建立原理和准则的过程，也是哲学探索的本质；智慧与前四个级别的

区别是它不束缚于历史，而是面向未来，即智慧是能够改变“将要发生”事情的知识。不论是商业智能还是人工智能，智慧的建立都离不开 DIKW 框架和数据分析的阶梯。

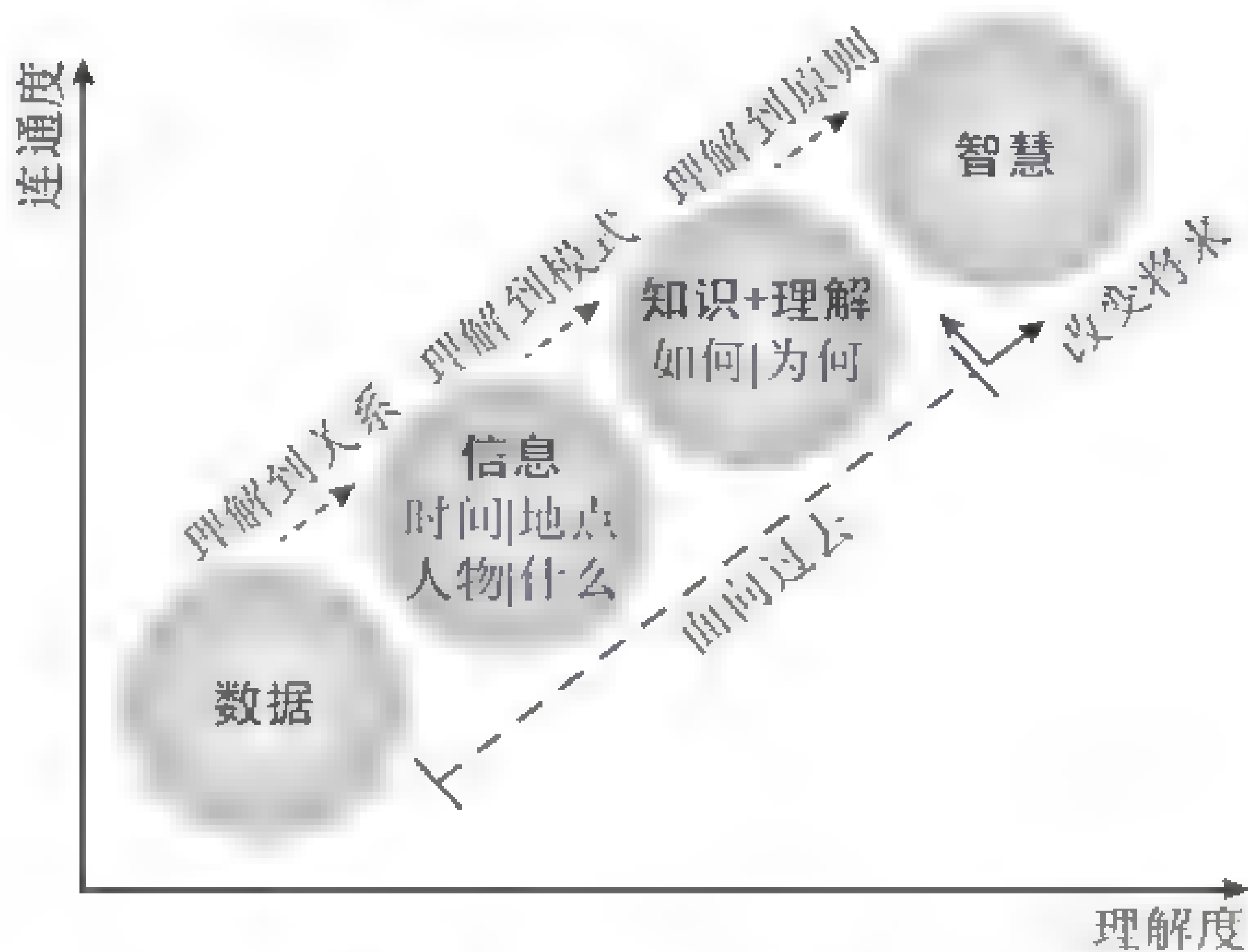


图 1-1 DIKW 模型

下面以 SAS 的 Hello World 程序（见程序 1-1）开始我们的 SAS 语言编程之旅。该程序只输出两行文本。SAS 与传统编程语言不同，默认它并不输出到操作系统的控制台窗口，而是 SAS 的日志窗口。

程序1-1 HelloWorld 程序

```
data _null_;
  put "THE POWER TO KNOW";
  put "-- Since 1976 --";
run;
```

1.1 语言概述

TIOBE 指数是用来反映某种编程语言的程度的指标，根据 2017 年 10 月最新的数据显示，SAS 编程语言占比 1.296%，排名第 21 位，而 2016 年 12 月的排名为第 22 位。编程语言本质上是人类用来与机器沟通，并在人类之间分享思维方法的工具，与它所需要解决的问题领域紧密相关。世界上没有哪一种语言能够解决所有问题，所以尽管计算机领域出现了超过成百上千种编程语言，但终究只有少数强大语言生存下来，SAS 便是其中的佼佼者，从 1976 年创立发展到现在已经 40 多年。

SAS 编程语言入门很快，但要精通需要较长时间，尤其是要掌握通用编程语言里面没有的一些 SAS 特性则需要花费较长的时间。SAS 作为数据分析领域特定的第四代编程语言（Fourth-Generation Programming Language，4GL），与广泛流行的第三代编程语言 C++、Java 和 C# 不同，它是专门为数据分析和报告处理中所涉及的复杂数据操作、图形图表制作、文档创建与输出而设计，不拘泥于通用计算机语言规范，而是以用户操纵数据、分析数据为根本导向。

SAS 语言总体上是面向过程的计算机语言，有传统编程语言的基本结构。但它不支持面向对象，而是以数据为导向。虽然从 SAS 9 开始引入若干系统预定义对象：哈希（Hash）、哈希遍历器（HIter）、JavaObj 对象和 ODSOUT 对象，并提供类似面向对象的成员调用语法，但用户至今不能在 SAS 语言传统的 DATA 步中创建自定义类；要使用准面向对象的语言结构需要在 SAS 的第二代 DATA（即 DS2）步中才可使用。然而，SAS 提供强大的互操作能力，用户可在 SAS 中调用 Java 对象和 Win32 API 函数来实现各种复杂功能，也可以在 SAS 代码中直接调用操作系统的各种命令。

SAS 语言中只有两种基本的数据类型：字符型和数值型，分别映射统计学中的定性数据和定量数据。在统计学的世界里，数据分析者并不关心数据的存储细节，而是关心数据的语义表达。从外部数据到 SAS 系统内部的数据存储表达之间，用户可以使用输入格式（INFORMAT）和输出格式（FORMAT）对数据进行读和写的格式化转换。

SAS 在一些过程中为这两种基本数据类型提供更丰富的数据类型支持，如从 SAS9.1 开始引入的函数编译过程步（PROC FCMP）提供类似 C 语言的结构体支持，而矩阵运算过程步 PROC IML 提供矩阵概念和专业运算。但总的说来 SAS 与通用编程语言所支持各种面向存储的基本数据类型：如布尔类型、整数类型、浮点类型和字符类型不同，它更关心的是统计学上的数据类型。

根据测量尺度划分，统计学上的数据类型只有定类（如性别）、定序（如年级）、定距（如摄氏温度）和定比（如重量）4 种基本数据类型，每种数据类型内所包含的信息量或熵值是不同的，因而适用的统计分析方法也就不同。

SAS 语言提供非常强大SAS宏语言预处理器，可实现SAS程序编译前的宏替换功能。这一特性允许 SAS 程序在编译和运行期间，动态改变程序代码自身，甚至可实现宏本身的递归调用。使用 SAS 语言的统计分析人员时常惊叹于 SAS 宏语言的强大功能，而一些不那么熟悉 SAS 宏语言的编程人员则时常为它与传统计算机编程语言的不同而困惑不解，以至于时常迷失在 SAS 宏与非宏的程序世界里。

SAS 语言是一门比较复杂的计算机语言，它到底是编译执行还是解释执行有时候连有经验的 SAS 开发人员也会感到困惑。鉴于 SAS 语言包含灵活的语言元素，SAS 代码中的 Macro 宏是由宏解释器展开的，但非宏的 DATA 步和 PROC 步则以步（Step）为单位由 SAS 依次进行编译执行。SAS 并不是逐句解释执行，而是按步编译执行。DATA Step 中包括的语句也分为编译阶段起作用的声明性（Declarative）语句和运行时起作用的可执行（Executable）语句两类。SAS 语言是兼具编译和解释的混合型计算机语言，鉴于此，维基百科的分类也很难处理是将 SAS 语言归入编译型语言（Compiled Language）还是解释型语言（Interpreted Language）。

SAS 语言最强大的能力是为分析编程人员提供了完备细致的数据访问，而不用太多考虑数据存储的格式和存储位置等细节，如 DATA 步和 PROC SQL 过程步就提供了各种各样的数据操作能力，而丰富的 PROC 步支持让分析人员专注于数据分析本身和参数设定。用户通常无须为各种标准化的严谨分析任务重新编写代码逻辑，SAS 已经为各种分析方法规范化了统一的形式和统计量，一旦 SAS 编程入门，通常只有不懂的统计方法和

类型而没有不会使用的 PROC 步。

从传统的编程语言转换为 SAS 编程语言进行编程，首先要谨记如下一些 SAS 语言的核心规则。

（1）SAS 程序由一系列 SAS 语句组成，所有的语句都以分号“;”结束。SAS 代码中可以嵌入待分析的数据行，但数据行不是 SAS 语句，因此它不需要以分号结尾。

（2）一个 SAS 语句可以跨多行编写，多个 SAS 语句也可以放在同一行上。SAS 语句可以从一行中的任意位置开始，代码缩进并非必需的。

（3）SAS 语句中的关键字以空格分隔的，通常由“关键字”或“关键字 – 参数”系列组成。某些语句在必选和可选选项之间会用斜杠“/”号进行分隔。

（4）SAS 语言元素不区分大小写，用户可以使用大写、小写以及它们的混合。但当字符串作为字符变量的数据时则区分大小写，如“Hello World”和“HELLO WORLD”是两个不同的字符串值。

（5）SAS 代码中标识符长度较短，用来标识数据库和外部文件的逻辑库引用（libref）和文件引用（fileref）名称最长不得超过 8 字节长度，而数据集（数据表）名称和变量（数据列）名称最长不得超过 32 字节长度。

（6）SAS 提供强大的 SAS 宏系统支持，含有百分号“%”或连字符“&”的代码文本会在编译前触发 SAS 宏展开。SAS 提供语言级别的宏支持，而不仅仅是简单的宏替换，它包括宏变量、宏函数、宏分支以及宏循环等完备流程控制。

总体上，SAS 程序主要由一系列的全局语句和 SAS 步构成，包括 DATA 步和 PROC 步。而所谓的 SAS 步则由一系列的 SAS 语句（Statements）构成。每一个 SAS 步都有开始和结束边界，SAS 根据步的边界进行独立编译和执行。图 1-2 展示了 SAS 代码的宏观构成。

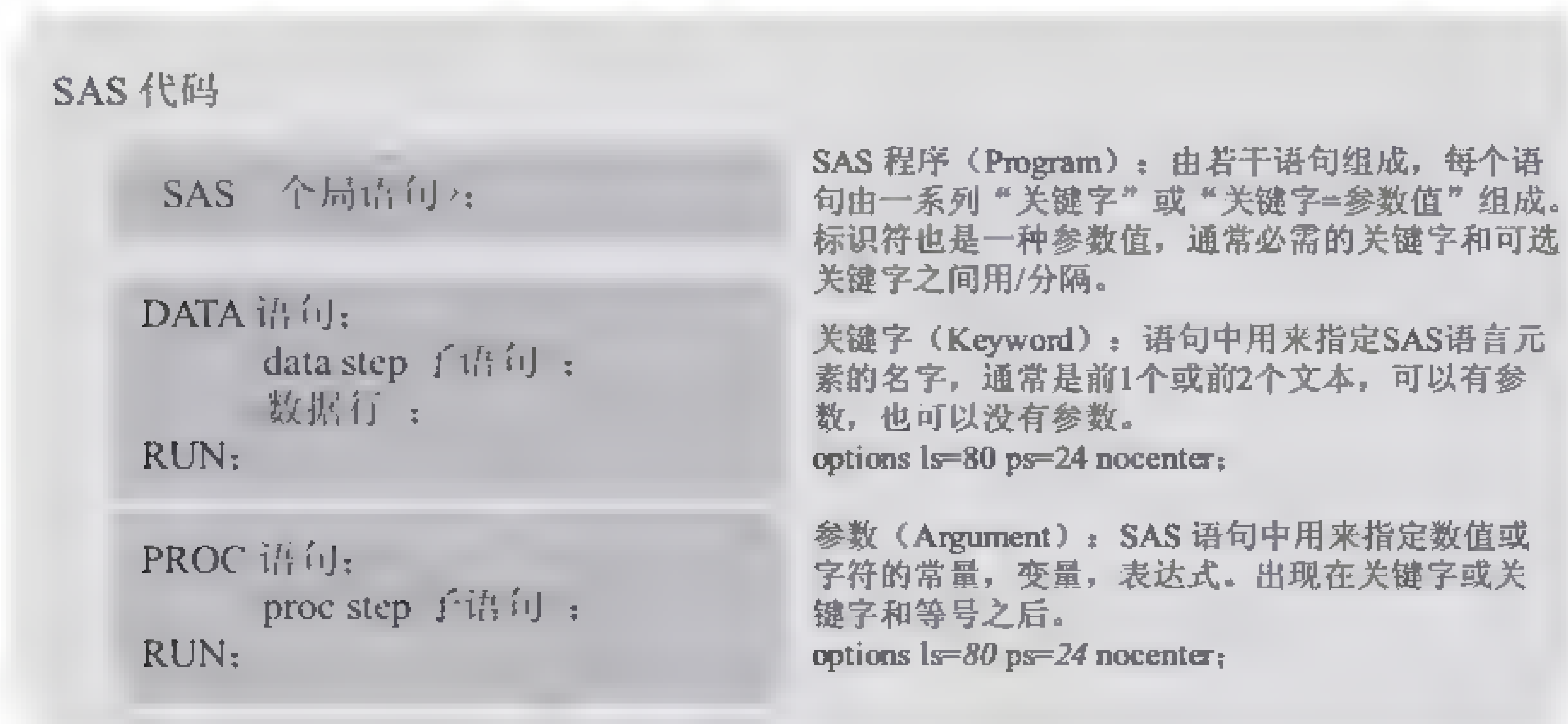


图 1-2　SAS 代码的宏观构成

SAS 步由 DATA 或 PROC 语句开始，默认结束于下一个 DATA 或 PROC 步的开始处。用户也可用 RUN 语句显式地标识 DATA 或 PROC 步的结束并提交执行之前的 SAS 代码。对于某些资源依赖型 PROC 语句，则通常需要以 QUIT 语句来提交当前 PROC 步并释放资源返回 SAS 会话中。比如 PROC SQL、PROC CAS 等过程步，这种过程步可包含多

个 RUN 语句，但只有在遇到 QUIT 语句时 SAS 才释放系统资源返回当前 SAS 会话。

- 全局语句：在 DATA 步或PROC 步之外，SAS还包括若干全局语句，通常用于指定全局选项或者其他全局性的功能 比如 TITLE语句就可用来指定后续PROC步输出报告的标题文字，其中TITLE#（#为数字，如 TITLE2）全局语句可用来指定特定级别的标题，最多可达11级 如果希望在输出的图表中关闭特定级别的报表标题，用户可以无参数调用 TITLE 语句来实现 比如程序 1-2 可设置报表输出的标题。

程序1-2 全局语句
```
title "The title of my first report"; /*设置输出报告的标题*/
title2 "Author: Yinliang Wu"; /*设置第二级的报告标题*/
title; /*关闭报告标题*/
```

各种 SAS 选项语句也属于全局语句，功能上类似于操作系统的环境变量，不过它用来指定当前 SAS 会话有关的系统设置，如程序 1-3 用于设置当前会话（Session）的语言区域属性（Locale）为英文，此时后续的过程步都会受此选项影响输出英文语言的报告。

程序1-3 全局语句
```
options locale=en_US;/*设置会话的语言区域属性*/
```

- DATA步：数据步负责为后续数据步或过程步准备待分析的数据，它是SAS语言核心的组成部分之一。其基本语法为

```
data mydata;
  <语句和数据>;
run;
```

比如程序 1-4 创建一行具有 5 列的数据表，其中 Name 和 Sex 是字符型变量，其他 3 个为数值型变量。它与系统数据集 sashelp.class 的表结构类似。

程序1-4 DATA步范例
```
data mydata;
  input Name $ Sex $ Age Height Weight;
  datalines;
LEON M 30 175 83.5
run;
```

- PROC 步：SAS过程步是执行特定任务的一系列SAS语句的集合，它以 PROC 语句开始，一般到下一个 RUN语句结束；如前所述，某些PROC如PROC SQL允许有多条RUN语句提交代码到DBMS内执行，但只有当该PROC最后一个QUIT语句运行后才会释放资源返回SAS会话环境 每个过程步都有自己特定的SAS 语句，也有很多过程步共享相同的SAS 语句和参数选项，如几乎所有的过程步都有data= 参数用来指定待处理的输入数据集名称（见程序1-5）。

```
程序1-5　PROC步范例
proc contents data mydata;
run;
proc print data-mydata;
  var name height;
run;
```

参数 data mydata 是 proc contents 和 proc print 两个过程步语句都有的选项，用来指定过程步的输入数据集。如果过程步没有指定 data= 参数，则系统默认使用当前 SAS 会话中最后使用或生成的那个数据集，该数据集的名称也存在于当前会话的系统宏变量 &SYSLAST。

- 程序注释：代码注释通常用于标注不可执行的文本，如描述程序的功能，或出于生成文档的目的在代码中添加说明性文本　注释还可用来在调试代码过程中将已经调试好的SAS代码暂时隔离，当代码运行时注释中的代码会被编译器自动忽略，但SAS注释依然会被写入SAS日志文件　鉴于SAS宏本质是文本替换，需要特别注意的一点是在MACRO宏代码中应尽量使用块注释，谨慎使用行注释以免导致不期望的宏展开，宏代码中使用行注释应中以 %* 开始，分号结束　SAS块注释和行注释如下

（1）块注释：SAS 语言支持 C/C++ 和 Java 等语言广泛使用的块注释，它以 /* 开始，以后续最近的 */ 号结束，注释可以包含分号以及任何长度的文本，也可以跨行（见程序 1-6）。但 SAS 代码不支持嵌套使用块注释。

```
程序1-6　块注释
/*
 * 块注释1…
 */
proc contents data=mydata;
run;
```

（2）行注释：行注释以星号“*”开始，结束于最近的一个分号“;”处。虽然它可注释多行文本，但它总是以最近的一个分号（包括引号中的分号）结束；其设计初衷用于调试过程中注释掉单行语句，跨多行的文本建议使用块注释以免产生意想不到的结果。（见程序 1-7）。

```
程序1-7　行注释
* 行注释1…;
proc contents data=mydata; * 行注释2…;
  * 行注释3…;
run;
```

虽然 SAS 代码在格式上具有很强的灵活性，但良好的代码风格能提高代码的可读性和可维护性。因此，一般情况下请遵循如下 SAS 代码格式化规范，使 SAS 代码具有较强的可读性。

（1）全局语句、DATA / PROC 步语句、步结束语句 RUN / QUIT 等语句应开始于第一列，而其他子语句通常采用逐级缩进，以显示层次结构关系。

（2）SAS 步与步之间通常用空行分隔，以表示 SAS 代码的编译边界，方便代码错误调试。

（3）当单行代码因参数较多导致长度较大时，应折行处理，并在该语句的结束分号后加上一个空行。

总的来说，SAS 的 DATA 步和 PROC 步是 SAS 语言对数据分析工作的精妙抽象和完美封装，数据步主要解决待分析的数据结构和数据准备问题，而过程步解决特定分析方法和流程的实现和封装；这两种 SAS 步就像数据结构和算法设计，大体上分别负责数据结构和算法逻辑实现。只有当需要更加复杂的自定义数据处理和分析算法时，才需要后面章节中将会讲到的各种函数封装进行扩展。

1.2 编程环境

SAS 编程需要使用什么样的开发环境？其实它跟其他计算机语言一样，可用任何纯文本编辑器编写 SAS 代码，如 Windows 平台的记事本，NotePad++ 或者 UltraEditor 工具。也可以使用 UNIX 上的 vi 来编辑代码，不过需要注意的是 Windows 平台使用回车换行符 CRLF 而 Unix 平台使用换行符 LF 进行文本换行。用文本编辑器生成的 SAS 代码文件，其文件编码（File Encoding）需要与 SAS 运行时的会话编码（Session Encoding）匹配，否则可能出现不期望的乱码或程序行为。这并不是 SAS 语言编程特定的文件处理问题，而是所有编程语言都会面临的源代码文件编码和编译器读取文件的所采用的编码之间的匹配问题。

在 SAS 执行环境中检查当前 SAS 会话编码，可使用 PROC OPTIONS 过程步检查（见程序 1-8），它会输出当前 SAS 系统使用的默认语言区域设置以及字符集编码信息。

程序1-8 检查当前SAS会话的Locale/Encoding 设置

```
proc options option=(locale encoding);
run;
```

对于用户用文本编辑器编写好的 SAS 代码，如何用命令行方式编译运行 SAS 代码？对于已经安装好 SAS 环境的机器，用户只需要调用 sas.exe 然后指定 -sysin 命令行参数将 SAS 代码文件的全路径传递给 SAS 即可。默认 SAS 代码运行后生成的日志文件会输出到当前路径，用户也可以使用命令行参数 -log 进行指定。比如：

```
C:\>"C:\Program Files\SASHome\SASFoundation\9.4\sas.exe"
    -sysin C:\temp\helloworld.sas -log C:\temp\helloworld.log
```

SAS 默认使用配置文件为 C:\Program Files\SASHome\SASFoundation\9.4\sasv9.cfg，也就是当前机器上安装 SAS 时生成的默认配置。检查该文件的内容可发现它默认指向了 SAS 安装目录下面的某个语言特定的配置文件。比如在中文环境上安装的 SAS，该 sasv9.cfg 文件的内容如下，表示默认使用中文配置来建立 SAS 会话环境并运行 SAS 代码。

```
config "C:\Program Files\SASHome\SASFoundation\9.4\nls\zh\sasv9.cfg"
```

如果用户想在中文的环境上运行特定配置的 SAS，如希望用纯英文版 SAS 执行用户代码，用户只需要在运行 SAS 代码时直接指定特定配置文件即可。检查该环境运行的 SAS 日志文件将会发现所有的输出内容变成了英文文本。

```
C:\>"C:\Program Files\SASHome\SASFoundation\9.4\sas.exe"
    -config "C:\Program Files\SASHome\SASFoundation\9.4\nls\en\sasv9.cfg"
    -sysin C:\temp\helloworld.sas -log C:\temp\helloworld.log
```

同理，如果想用英文版 SAS 执行代码但程序中又要支持处理中文数据，用户则可以使用 SAS 安装目录中的 nls\1d\sasv9.cfg 配置文件；如果想用 Unicode 版本的 SAS 来运行代码，则需要使用 nls\u8\sasv9.cfg 配置文件。

图形用户界面

由于 SAS 被设计用于数据分析，因此更常见的情况是利用图形用户界面（GUI）来编写 SAS 代码并调试运行。只有开发完毕后才用上面提到的命令行方式包装到后台静默执行或者作为操作系统服务运行。到目前为止开发 SAS 程序，通常在如下 4 种环境之一进行。

（1）Base SAS 运行环境：在机器上单独安装 SAS Foundation 软件，然后在 SAS 图形界面中进行编程。这是 SAS 最传统和最常见的 SAS 编程环境。

（2）SAS Enterprise Guide（简称 EG）Windows 客户端环境：它是运行在 Windows 平台比 Base SAS 更高级的 SAS 企业客户端。用户即使不熟悉 SAS 编程语言，也能够用鼠标拖拽的方式利用 EG 快速进行可视化编程。EG 是独立于 SAS 运行环境的一个客户端产品，它需要调用本地安装的 Base SAS 运行环境，或者连接到远程安装的 SAS 环境来执行生成的 SAS 代码，远程的 SAS 环境可以是 SAS Workspace Server 或 SAS Stored Process Server 服务器，EG 客户端使用 SAS Integration Technologies 的集成对象模型（IOM）与后台进行通信。

（3）SAS Studio 浏览器客户端环境：它是基于浏览器的 Web 客户端，让用户可以在本机不安装 SAS 和 Enteprise Guide 等客户端的情况下进行 SAS 编程。SAS Studio 中提交的代码通过 SAS 互联网基础设施平台（WIP）或 Viya 组件运行在远程服务器上，远程服务器处理完毕，结果会返回并显示在 SAS Studio 浏览器窗口。SAS Studio 产品使 SAS 编程环境可以部署在云端，而客户端可实现零安装部署就能进行 SAS 编程和调试。比如，SAS 为学术界免费提供的云服务 SODA，用户只需注册即可实现随时随地进行 SAS 编程和调试。

（4）SAS University Edition 虚拟机环境：即所谓的 SAS 大学版，实际上它是 SAS 免费提供的虚拟机版本，其中不仅包括 SAS Studio 和 Base SAS 基本环境模块，还包括 SAS/STAT、SAS/IML、SAS/ETS 和 SAS/ACCESS Interface to PC File 等组件，主要用于 SAS 编程的学习培训。用户只需要从 SAS 官网 https://www.sas.com/zh_cn/software/university-edition.html 下载虚拟机即可。另外，用户也可以在亚马逊 AWS 网站上创建亚马逊账号，然后在“Marketplace”中输入“SAS University Edition”找到 SAS 大学版，随后按照提示启动编程界面。亚马逊的 AWSMarketplace 网址为 https://aws.amazon.com/marketplace。

由于数据分析系统的特殊性，SAS 编程环境界面大体包含以下几个基本窗口：

- 编写/提交 SAS 代码的[程序]窗口或 [代码] 窗口；
- 查看SAS 代码运行细节的 [日志] 窗口；
- 显示输出结果的 [输出] 窗口。

比如在 Base SAS 的代码窗口中输入前面的 HelloWorld 程序，点击菜单 [运行 (R)] → [提交 (S)] 就可看到程序运行结果，用户也可以直接点击工具栏上的运行按钮，或直接按 F3 快捷键提交代码运行，这样就可在日志窗口中看到提交代码的执行细节和结果。

- Base SAS环境（见图1-3）。



图 1-3　Base SAS 环境

- SAS Enterprise Guide 环境（见图1-4）。

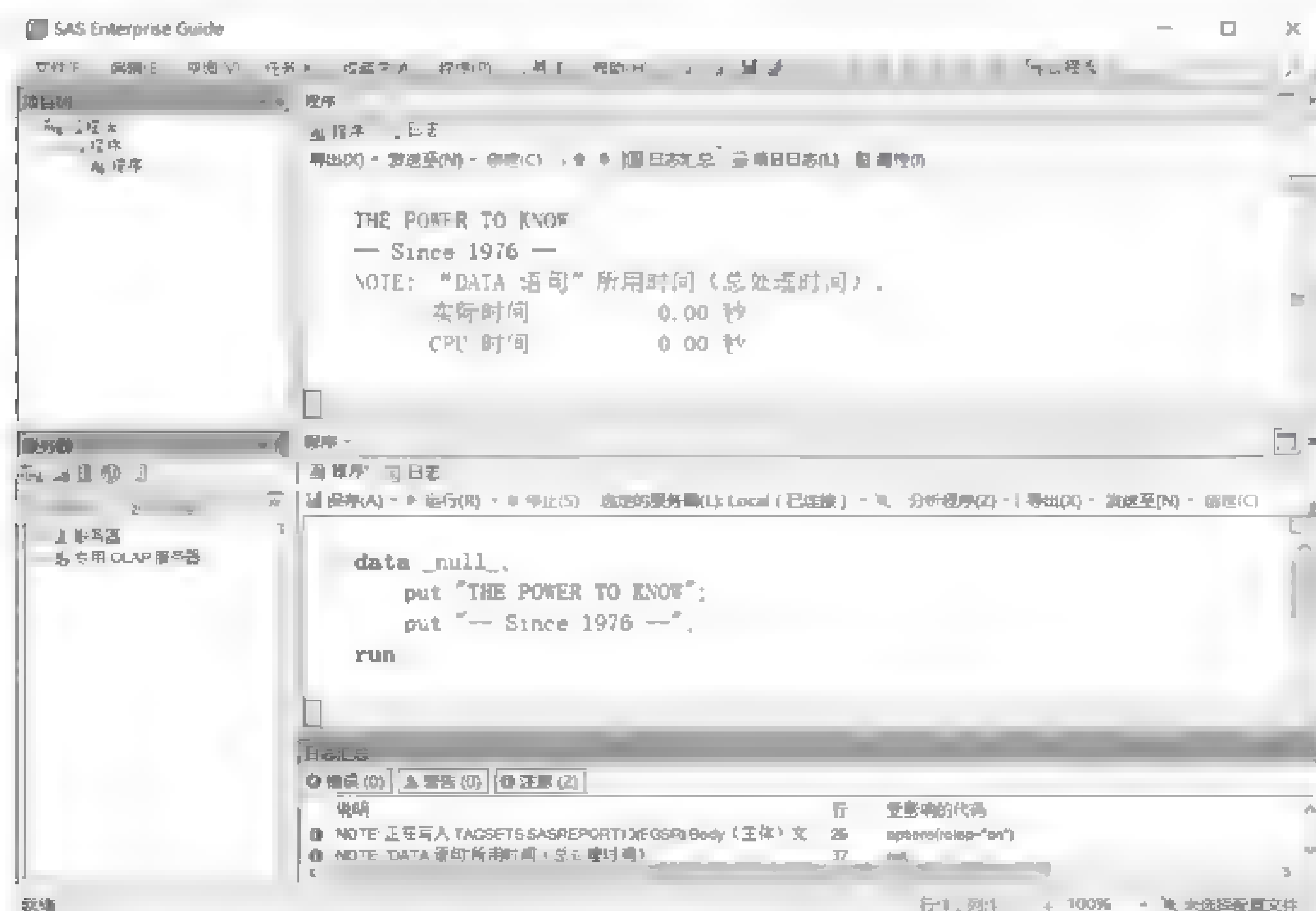


图 1-4　SAS Enterprise Guide 环境

- SAS Studio环境（见图1-5）。



图 1-5 SAS Studio 环境

当 SAS 代码提交编译运行时，SAS 会自动检查语法错误，如果发现语法错误，错误信息会输出到 SAS 日志窗口。语句中常见语法错误包括关键字拼写错误、无效选项、引号不匹配，以及语句结束符分号缺失等问题。

SAS 语法错误分为 ERROR 和 WARNING 两大类，分别以红色和绿色显示。红色的信息包括错误所在的行和列，以及对错误的详细描述；红色的错误需要用户仔细检查并加以修正，而绿色警告不妨碍后续程序的运行。

由于 SAS 编译器具有强大的拼写错误容错能力，它能够智能修正的错误被归入 WARNING 类，代码依然能正常运行。而红色表示 SAS 未能修正的错误，用户必须修正后才能正常执行。比如程序 1-9 示例代码依然能正常运行，尽管 PROC PRINT DATA 等几个关键字都被拼写错了（多了个 X），其仍能输出如图 1-6 所示结果。

程序1-9 检查SAS 代码容错能力

```
procx printx datax=sashelp.class;
run;
```



图 1-6　语法容错能力

对于一些参数错误，如指定了不存在的数据集 sashelp.classx，SAS 则会报告 ERROR 错误。

```
59   proc print data=sashelp.classx;
ERROR: 文件"SASHELP.CLASSX.DATA"不存在。
60   run;
```

很多时候正是因为 SAS 运行环境对代码的容错能力太强，一些错误会隐藏得较深，不能被用户立即发现。因此用户可采用分块调试的办法将语法错误限制在最小的范围内。比如下面一行代码，SAS 在第三次提交的时候才会报告错误。

```
title "Hello World;
```

SAS 程序主要由数据步（DATA Step）和过程步（PROC Step）组成，SAS 编译器会按步编译运行。如果没有语法错误则会启动编译与执行，重复这个过程直到所有的 SAS 程序"步"被处理完毕。

当代码编辑窗口被激活时用户可以随时点击菜单 [文件 (F)] → [保存 (S)] 来保存你的代码。SAS 使用默认文件编码保存程序，如在中文 SAS 环境上默认使用的是 Simplified Chinese （EUC）编码。用户也可以在保存代码对话框中指定 Unicode UTF-8、 Unicode（UTF-16LE）或 Unicode（UTF-16BE）编码格式进行保存，同时也可指定是否输出字节顺序标记（Byte Order Mark， BOM）进行存储。

字节顺序标记 BOM 就是用于告诉文件处理程序在读取文件内容时，该目标文件的正确编码。BOM 约定对进制文件最开始的 2 字节或 3 字节进行特殊处理，如果某个文件的前 3 个字节为十六进制值的 EF BB BF，则表明该文件为 UTF-8 有 BOM 的编码格式文件；如果前 2 个字节的十六进制为 FF FE 或 FE FF，则分别表示文件为 UTF-16 Little Endian 或 UTF-16 Big Endian 编码格式。

对于 Hello World 程序，你可能会有疑问——为什么有输出窗口却没有任何输出信息？ SAS 程序的输入输出与传统编程语言的文件 I/O 不同，SAS 的输出窗口是给前面提到的 SAS ODS 输出交付系统的 LISTING 目标使用的。由于 Hello World 程序没有任何 ODS 输出，只调用了 PUT 语句输出日志，因此输出窗口没有结果。程序 1-10 尝试创建一个 SAS 数据集，然后将该数据集打印到输出窗口。

程序1-10　建立数据集并输出到结果窗口

```
title "My data";
data mydata;
  Name="Yinliang";
  Sex='M';
  Age=30;
  Height=175;
  Weight=83.5;
run;

ods _all_ close; *关闭所有的ODS 目标;
ods listing; *打开 ODS Listing 目标用于输出;
proc print data=mydata;
run;
```

提交代码执行后，SAS 首先创建数据集 WORK.MYDATA. WORK 是 SAS 运行默认的 SAS 逻辑库。在输出窗口（见图 1-7）SAS 会打印出用户生成的数据集 WORK. MYDATA。

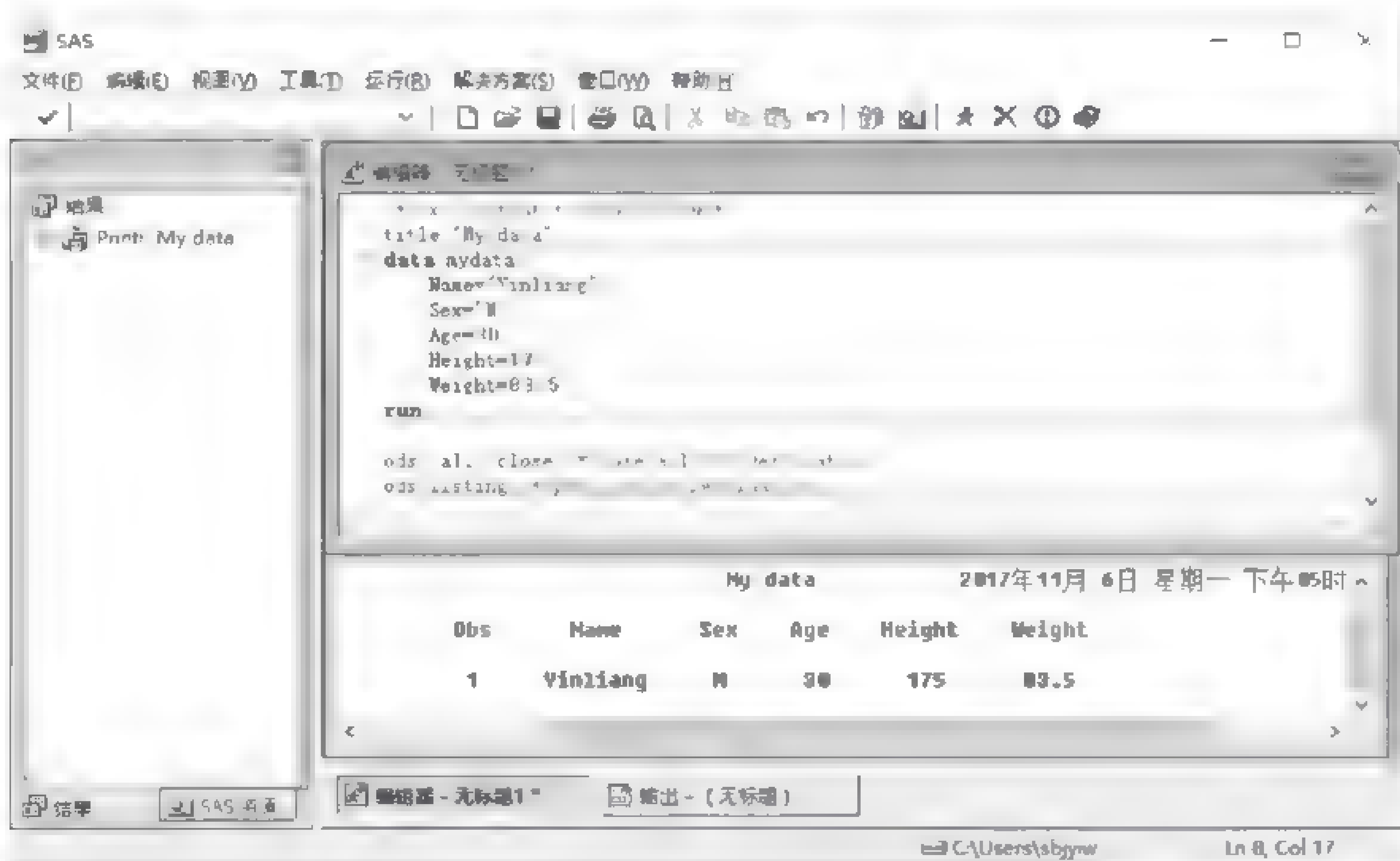


图 1-7　WORK.MYDATA 内容

需要查看 SAS 数据集的数据结构和定义时，可以使用 PROC CONTENTS 过程步进行查看。

```
proc contents data mydata;
run;
```

执行结果将显示在输出窗口（见图1-8）。可以看到数据集编码为“euc-cn Simplified Chinese （EUC）”，数据表示法为 Windows 64，表示数据是在 Windows 64 位的平台上生成。在与引擎/主机相关的信息中，文件名是数据集在磁盘上的存储路径。最后一节则是数据集中所有变量的类型和长度，可以看到包括数值和字符两种类型，数值的默认长度为 8 字节。



图 1-8　数据集的元数据信息

Base SAS 需要在单机环境或者服务器环境下单独安装，下一节将展示如何实现不安装任何 SAS 运行环境，就可在 Web 浏览器中随时随地编写自己的 SAS 代码。

1.3　SAS Studio 编程

SAS 软件具有巨大的商业价值，一般的人很难获得最新的安装拷贝。读者该如何学习 SAS 编程呢？ SAS 考虑到全球不断增长的数据分析需求以及体现对学术研究领域的一贯支持，SAS 提供基于 SAS 私有云的 SAS Studio 应用服务 SODA。

SODA 全称 SAS® OnDemand for Academics 是 SAS 为学术群体提供的免费在线应用服务环境。运行在 SODA 上的 SAS Studio 可以让你在任何时间、任何地点编写/运行自己的 SAS 分析代码，而且所有用到的数据和代码都会存储在 SAS 私有云。这是目前除了在本地安装强大的 SAS 系统外，学习使用 SAS 程序开发成本最低和最快捷的方式，用户只需要一个电子邮箱账号和浏览器即可。

（1）在浏览器中输入网址 https://odamid.oda.sas.com/SASODARegistration 访问

SAS® OnDemand for Academics（见图 1-9），按要求提交必要信息注册 SODA 账号，你的邮箱将会收到用户 ID，该用户 ID 将会在后面用来登录 SODA 控制中心来使用 SODA 提供的应用服务。

图 1-9　注册 SODA 账号界面

（2）注册完成后，通过网址 https://odamid.oda.sas.com 访问 SODA 控制中心。注册完成后，用户并不能马上收到注册信息，需要耐心地等待一段时间。登录 SODA 控制中心时，需要使用邮件中收到的【用户 ID】和【用户密码】，其中【用户 ID】并不是邮箱地址 yinliangwu@gmail.com，而是 SODA 分配的用户 ID，如 yinliangwu0。SODA 登录界面如图 1-10 所示。

图 1-10　登录 SODA 界面

登录后，系统将引导用户到 SAS ODA 的控制中心（见图 1-11）。页面上用户会看到一个 SAS® Studio 应用，点击它即可启动 SAS® Studio 应用。

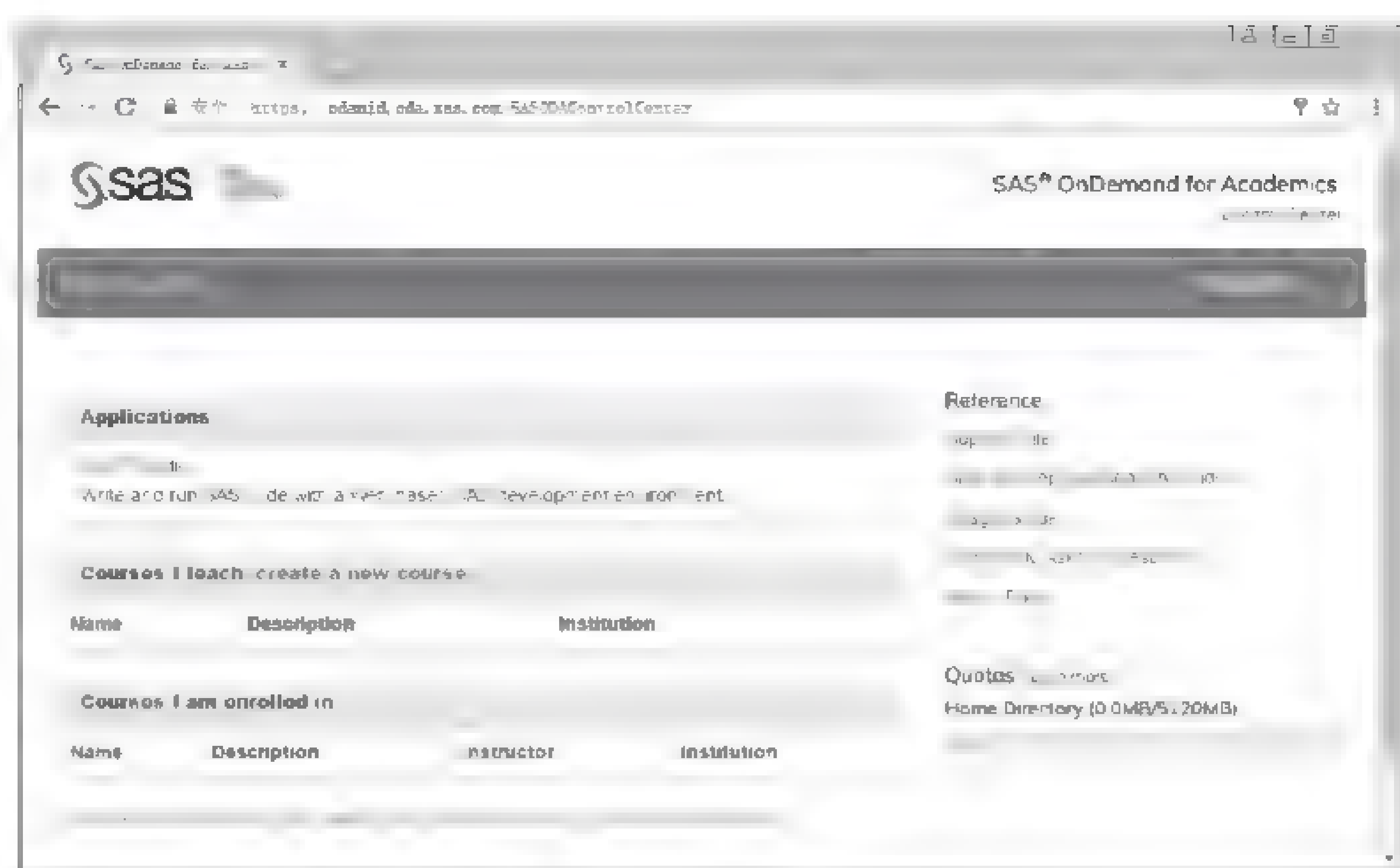


图 1-11　SODA 面板

用户也可在登录 SAS ODA 控制中心后，直接在浏览器中输入如下网址访问 SAS Studio 应用 https://odamid.oda.sas.com/SASStudio。目前在 SODA 上提供服务的 SAS Studio 为 3.5 企业版，它支持 Chrome 27+、IE9/IE11、Firefox 21+ 和 Apple Safari 6.0+ 以上版本的浏览器。后台 SAS 服务器为 Linux 64 位版本的 SAS、SAS Studio 为用户提供了一个编写和远程执行 SAS 程序的高效 Web 开发环境。

（3）SAS Studio 主界面包括顶部的菜单栏、左侧的导航面板和右侧的内容窗口（见图 1-12）。导航面板包括“服务器文件和文件夹”“任务和实用程序”“代码段”“逻辑库”“文件快捷方式”和“SAS 文件夹”等模块。其中“服务器文件和文件夹”和“逻辑库”比较常用，“服务器文件和文件夹”是用户在云端服务器上的磁盘存储空间，一般映射到用户特定的主目录 /home/<userid>，如 /home/yinliangwu0。

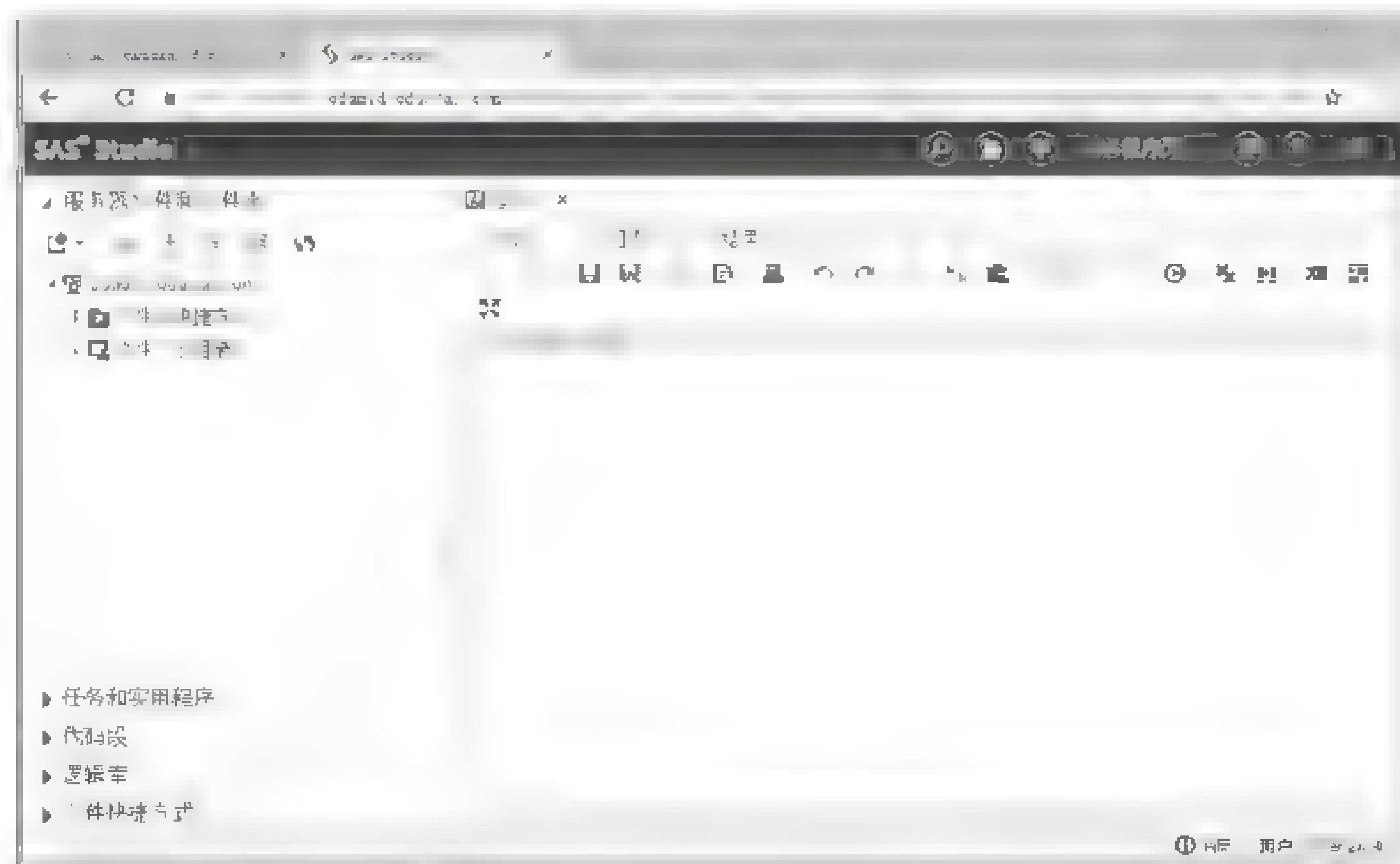


图 1-12　SAS Studio 主界面

右侧提供了程序窗口（见图 1-13），包括“代码”“日志”和“结果”3 个窗口。一般步骤是在代码窗口中输入 SAS 程序，然后点击代码页签工具栏最左边的运行按钮，或者直接按 F3 功能键直接执行代码。比如：

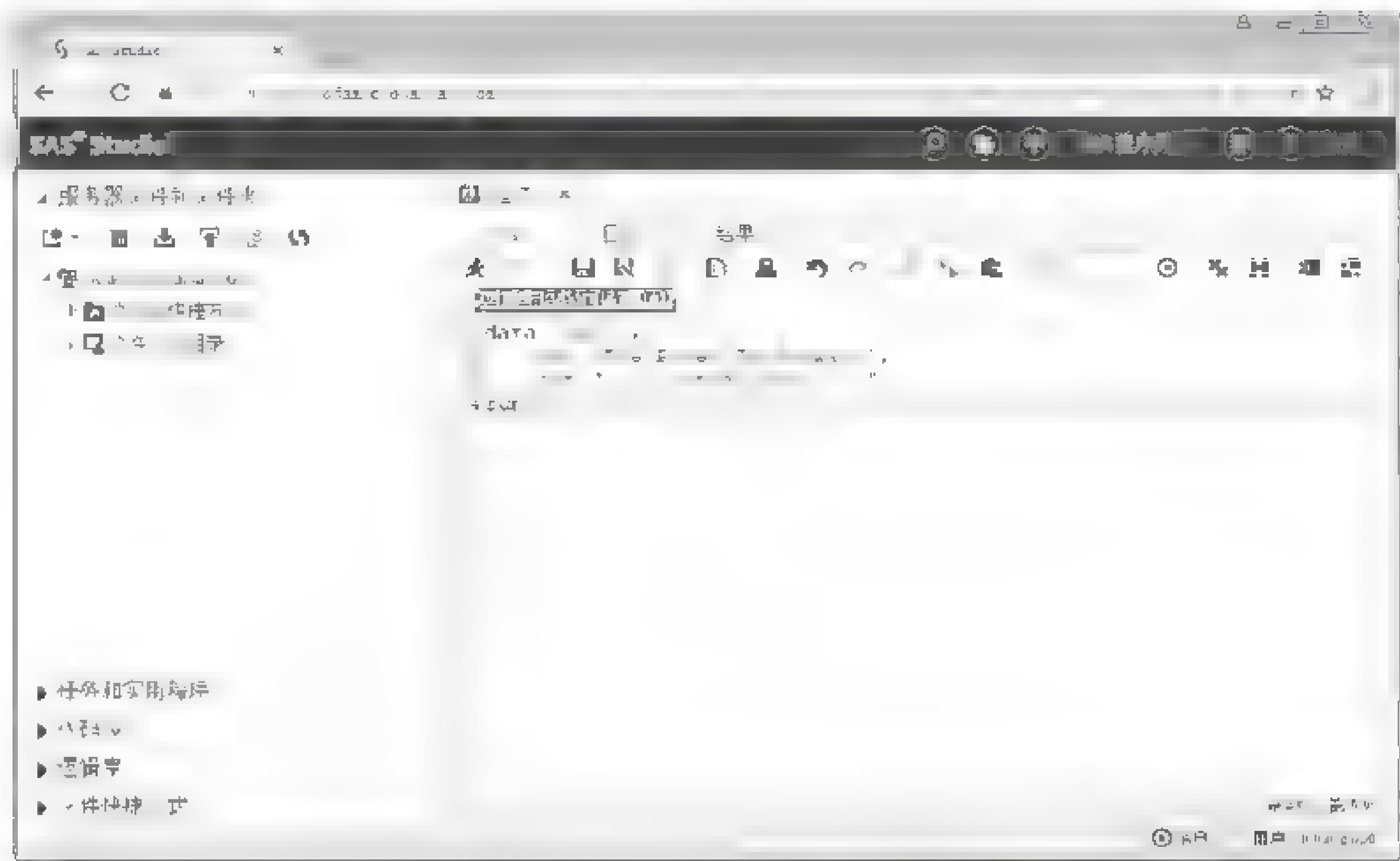


图 1-13　SAS Studio 程序窗口

执行日志可以查看 SAS Studio 日志窗口（见图 1-14），包括提交的源代码信息。



图 1-14　SAS Studio 日志窗口

结果窗口是用来输出 ODS 结果的，用户可以在程序中输入如下代码，然后按 F3 功能键运行 SAS 代码即可看到 ODS 输出结果。在 SAS 中一般 PROC 步会将分析的结果输出到 ODS 目标（见图 1-15）。

```
proc print data=sashelp.class;
run;
```

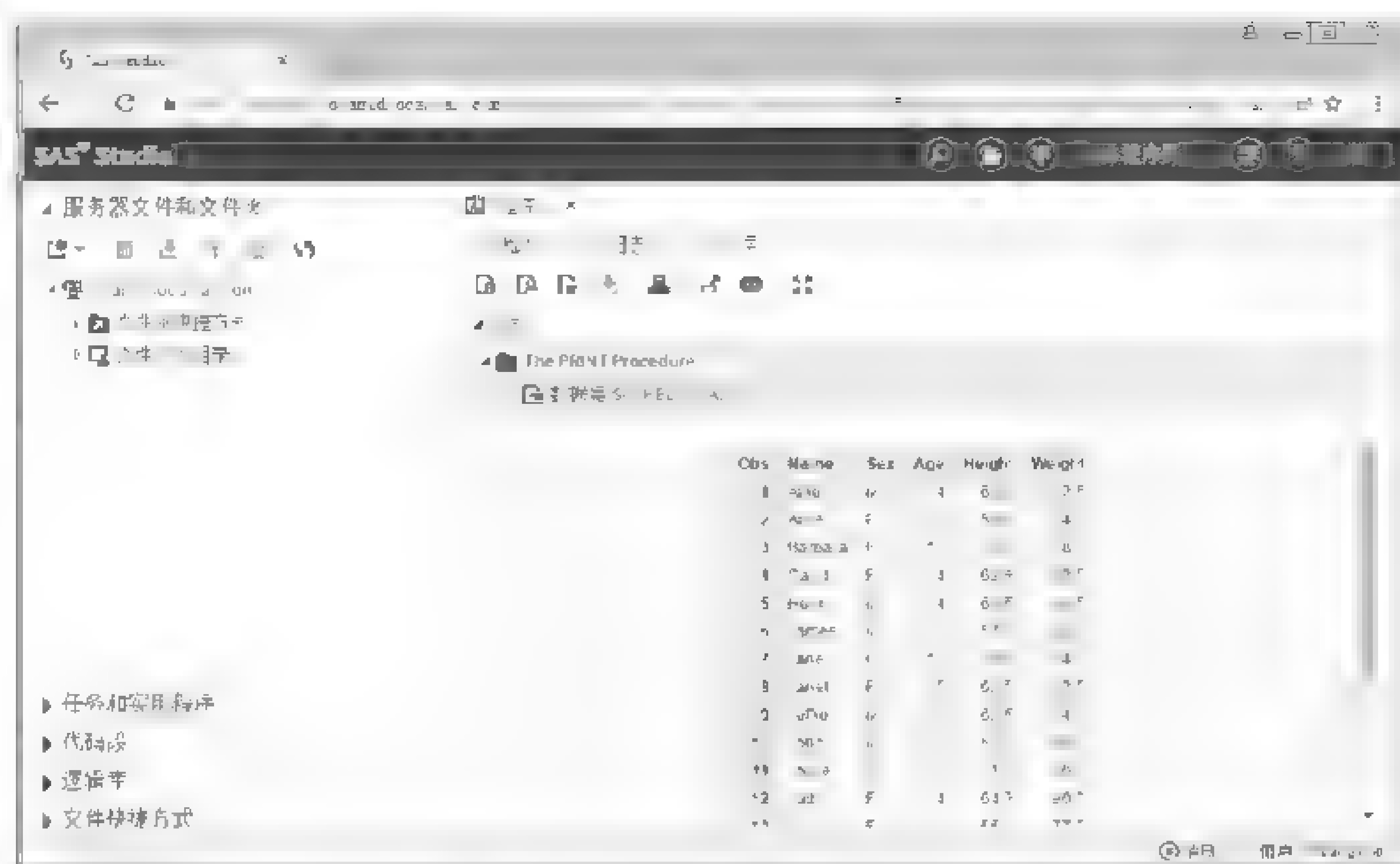


图 1-15　SAS Studio 结果窗口

在分析中用户需要上传自己的数据文件，可以点击左侧导航面板中的“文件（主目录）”，然后在菜单中选择“上载文件……”（见图 1-16）。

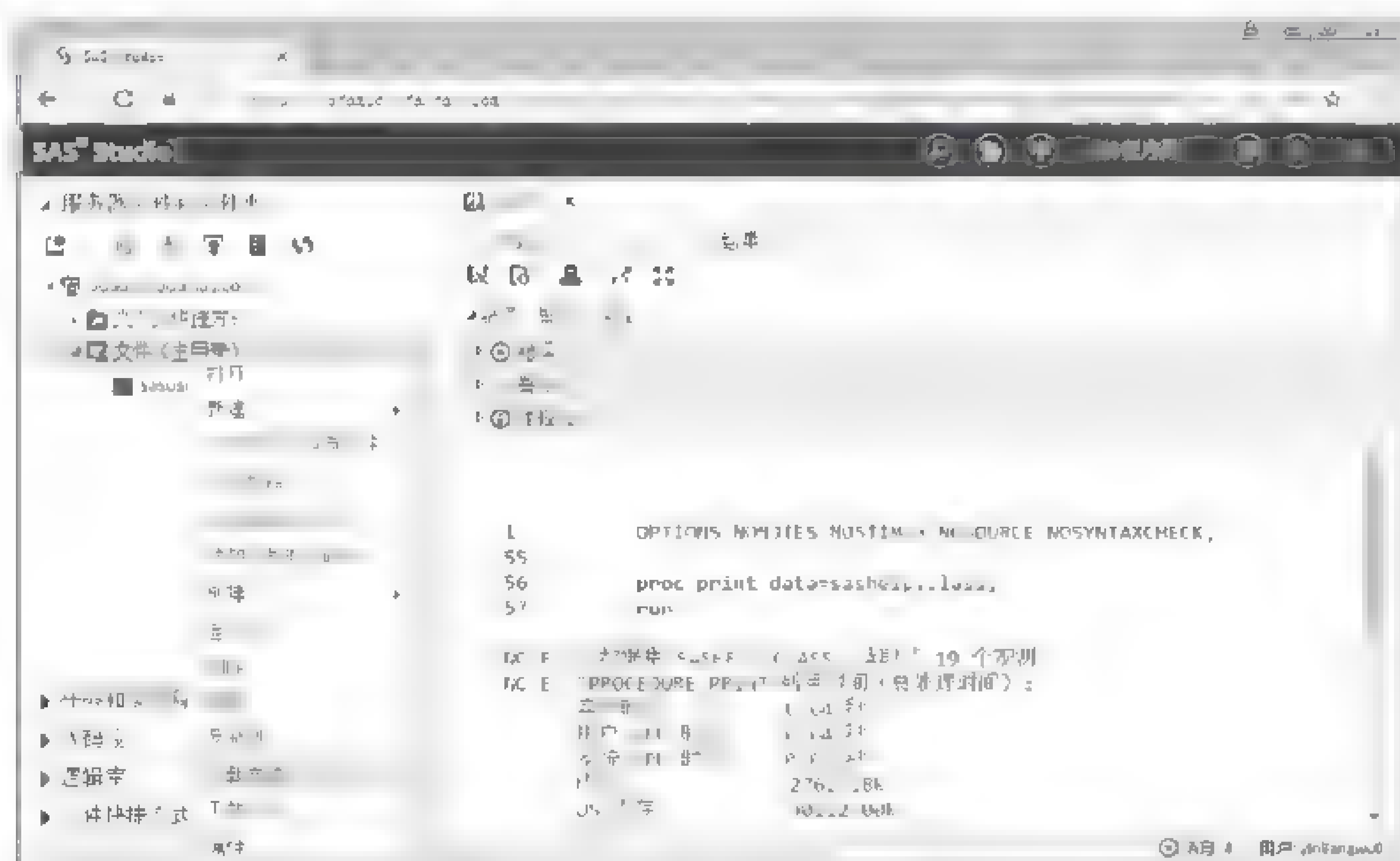


图 1-16　上载文件到主目录

在弹出对话框（见图 1-17）中单击“选择文件”，然后选择本地计算机上的数据文件上传到远程服务器，如 C:\temp\class.csv。

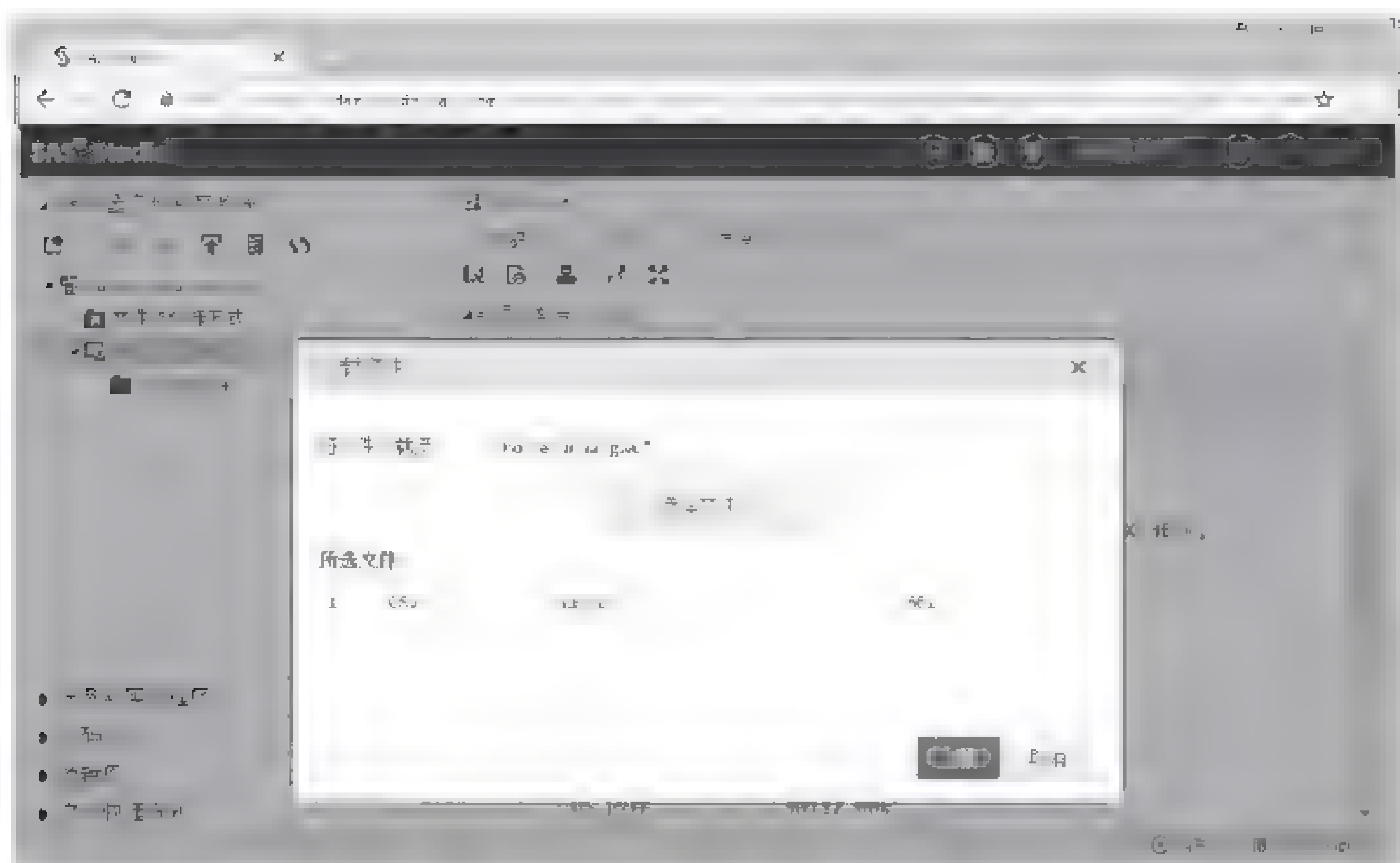


图 1-17　“上载文件”对话框

上传完毕在左侧的导航面板“文件（主目录）”中将出现 class.csv 文件，表明该文件已经上传到云端。如果要将该 csv 数据导入 SAS 系统，用户可以双击它，SAS Studio 会自动生成导入数据所需的 SAS Code，用户也可以选择不同的参数进行数据导入。

云端的 SAS 环境为了让全球所有用户都能使用、支持所有语言和国家地区的用户，其默认数据文件被设置为 UTF-8 编码。但在中文 Windows 环境下 SAS 生成的文本文件默认是 GB2312/GBK 编码，用户可以使用记事本的“另存为”功能，将文件转存为 UTF-8 编码格式即可完成转码工作。比如，如果用户上传一个本地 Windows 导出的 csv 文件，而在 SAS 导入的时候会默认上传的文件是 UTF-8 编码格式，此时就会出现乱码。解决方法如下所述。

（1）使用记事本或其他文本工具（如 Notepad++ 或 Ultra Editor）将数据文件打开，然后使用“另存为”，选择 UTF-8 编码保存即可进行转码。

（2）用户也可在数据导入的 SAS 代码中明确指定待输入文件的正确编码方式，由 SAS Proc Import 进行编码转换，将数据正确导入 SAS 数据集，其默认编码为 UTF-8。

如果用户上传的是使用 Windows 导出的 csv 文件，用户需要在将该文件导入时明确指定该输入文件为 GB2312 编码，步骤如下：双击 class.csv，然后在右侧 class 页签下点击“编辑”，SAS Studio 会自动生成对应的 SAS 代码 “程序 2”（见图 1-18），修改 FILENAME 语句如下，运行即可正确导入数据。

```
filename reffile'/home/yinliangwu0/class.csv' encoding=gb2312;
```

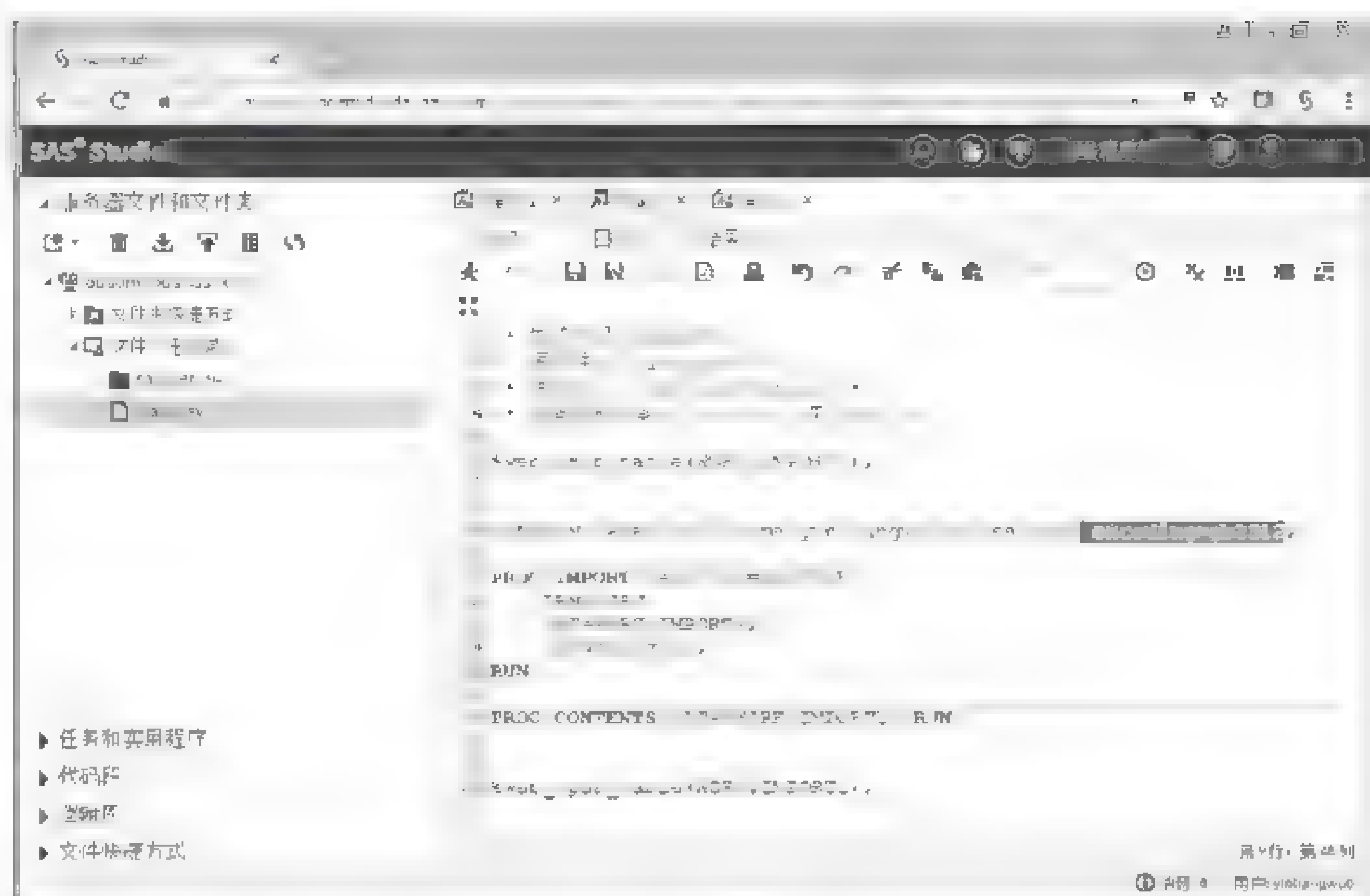


图 1-18 设置导入数据的文件编码

在结果窗口中可以看到导入完毕的数据，如 WORK.IMPORT1。在缺省情况下输出数据都会放到临时逻辑库 WORK。此时调用 PROC CONTENTS 输出的数据集的元数据信息如图 1-19 所示。



图 1-19 数据集的元数据信息

调用 PROC PRINT 打印数据集内容信息如图 1-20 所示。

用户可以通过鼠标右键点击左侧导航栏下的“逻辑库 / 我的逻辑库 /WORK/IMPORT1”来更改数据集名称，如改为 WORK.CLASS（见图 1-21）。

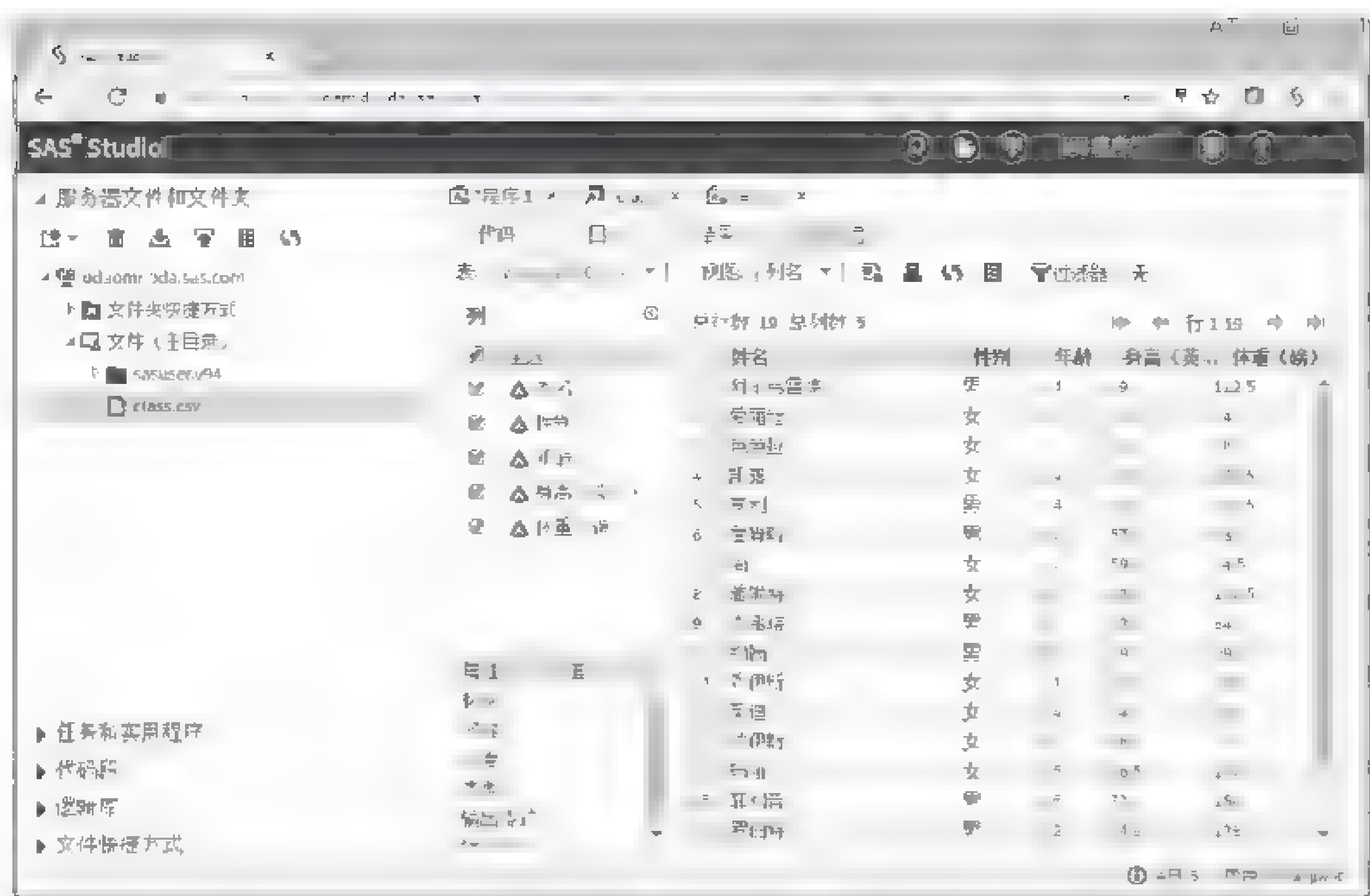


图 1-20　数据集的数据内容

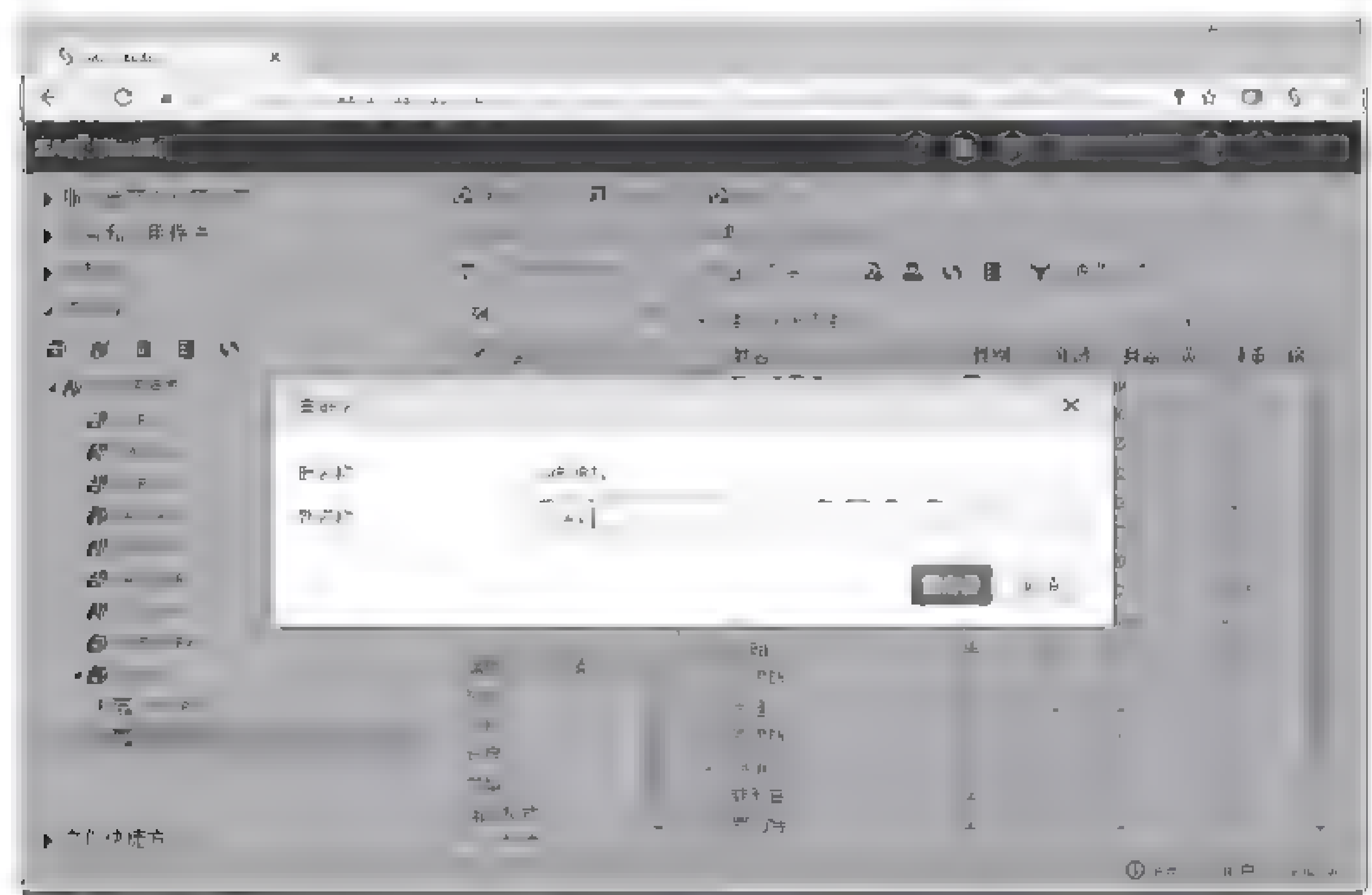


图 1-21　更改数据集名称

这样，用户就可以在本次 SAS 会话期间，即在本次登录 SAS Studio 直到关闭浏览器退出 SAS Studio 应用期间一直使用 WORK.CLASS 数据集，修改代码如下，即可查看导入的数据（见图 1-22）。

```
proc print data=work.class;
run;
```

运行代码用户将看到打印的数据变为刚上传后导入的那个数据集 WORK.CLASS。

用户也可将自己的 SAS 源代码保存到云端的服务器，如用户选择保存到“文件（主目录）”，文件名为 PrintClass.sas 即可（见图 1-23）。

保存到服务器上的文件 /home/yinliangwu0/PrintClass.sas，从窗口下方的状态栏中可以看到路径（见图 1-24）。

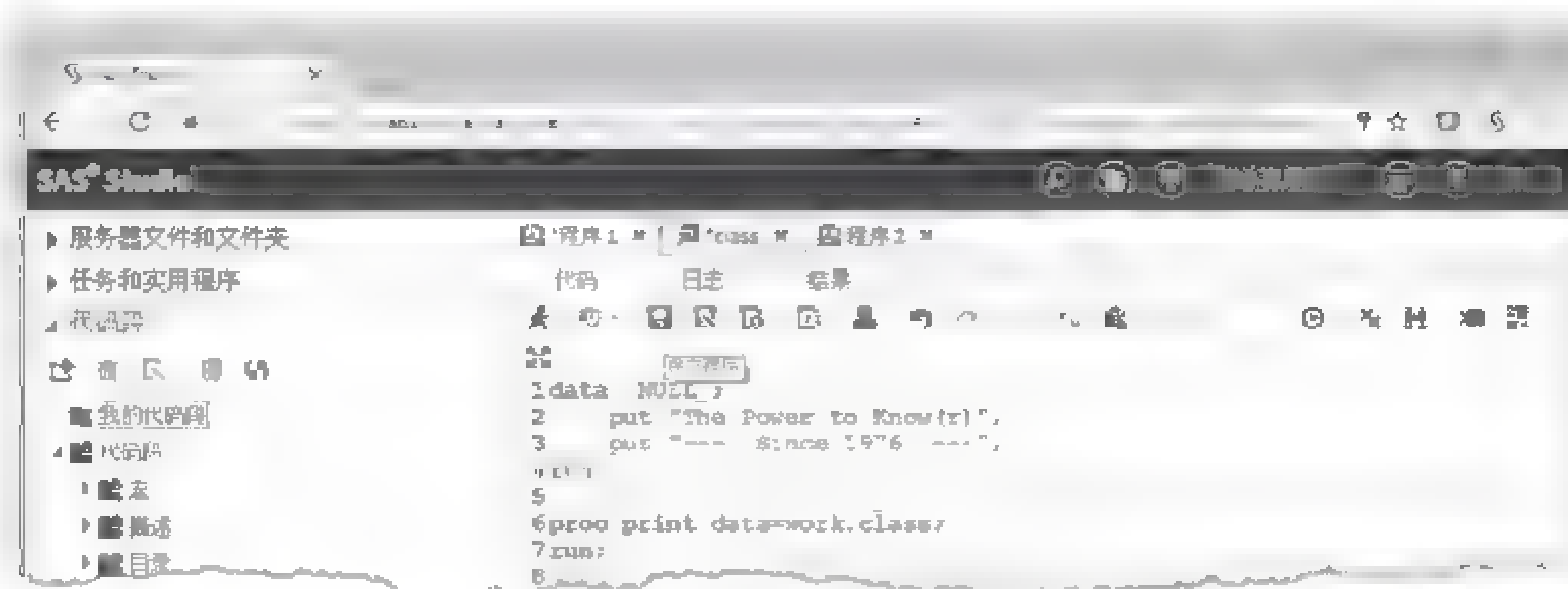


图 1-22　引用临时数据集

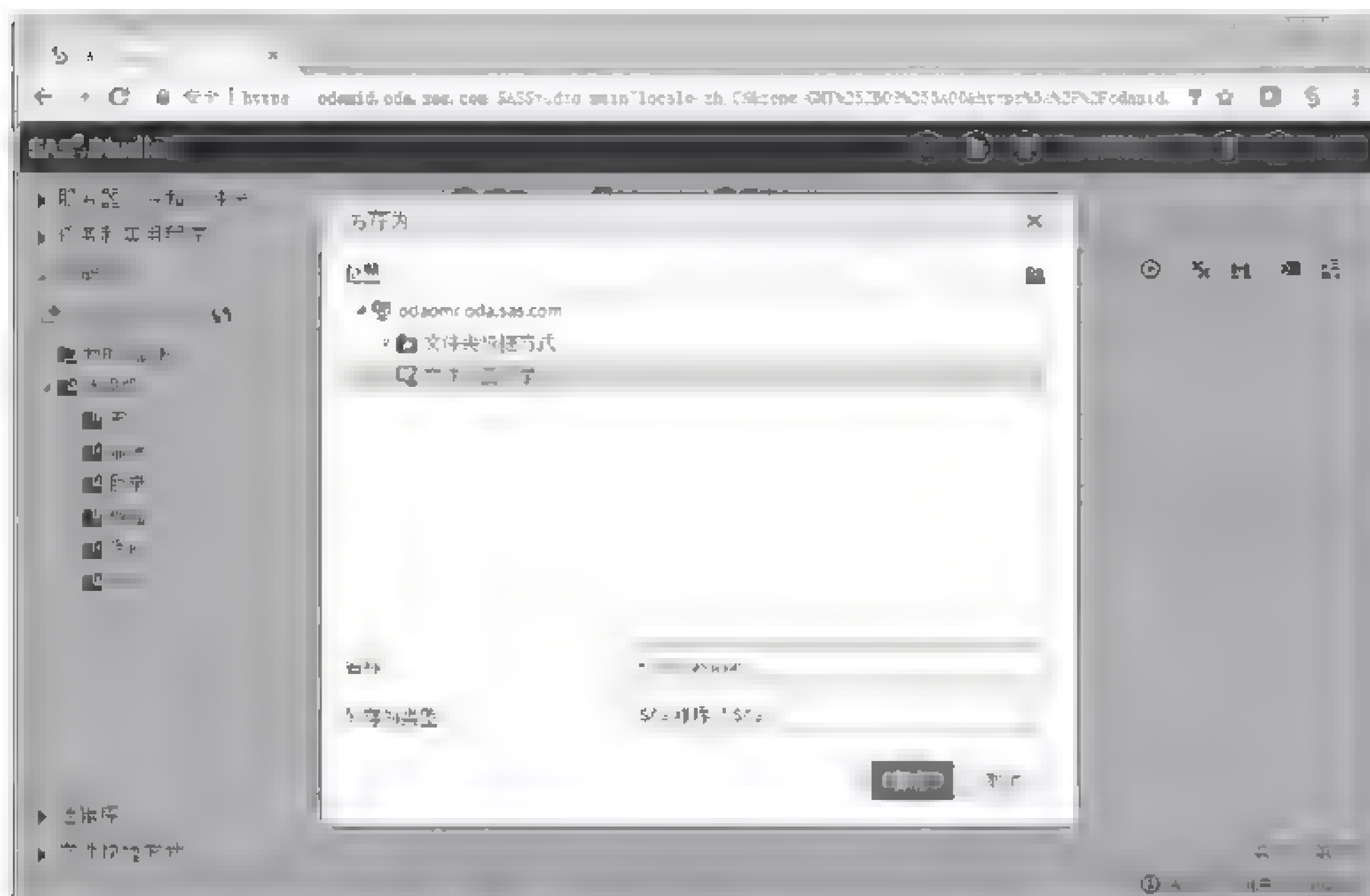


图 1-23　保存 SAS 源代码到服务器

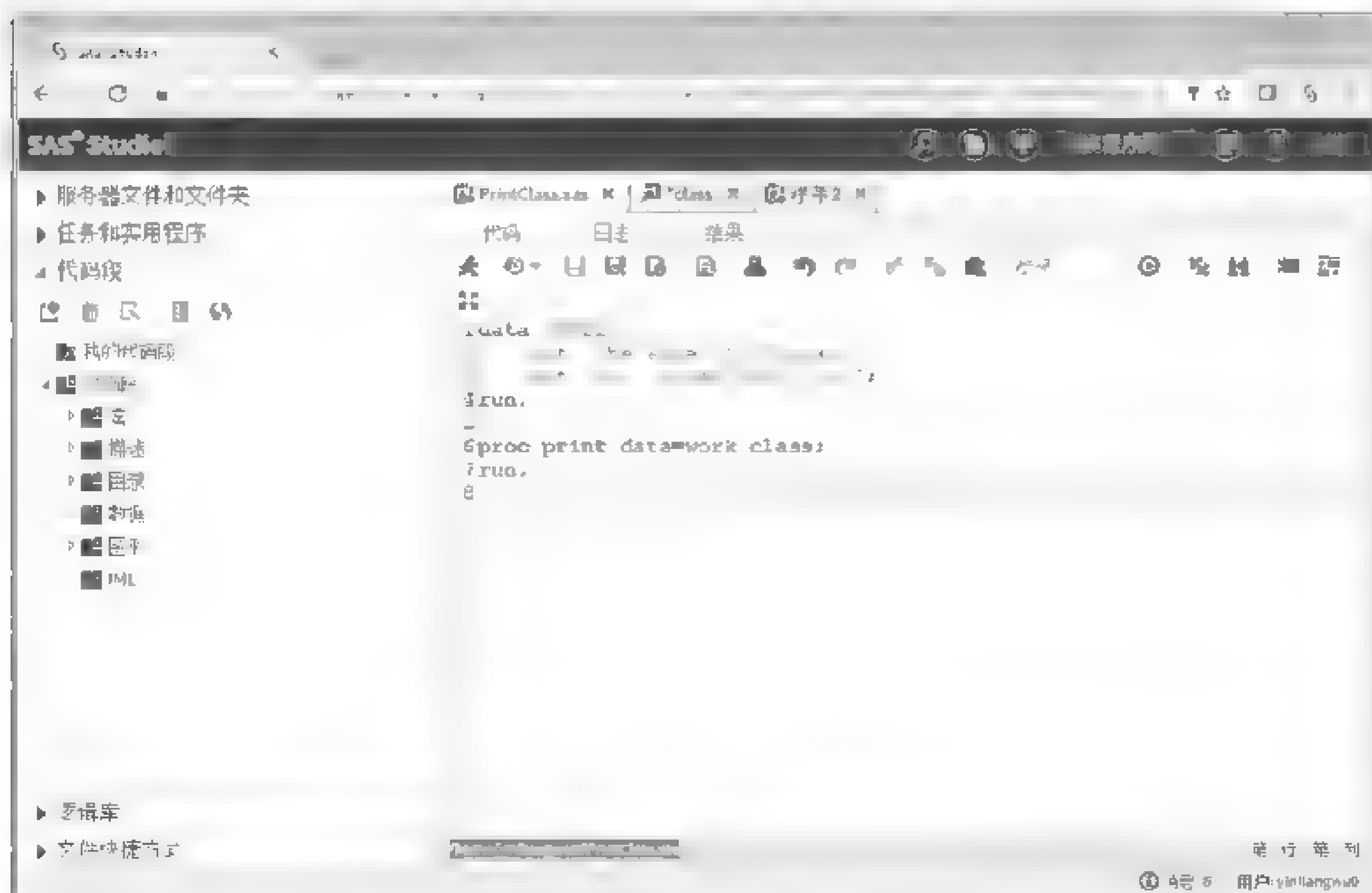


图 1-24　SAS 源代码在服务器上的路径

由于 WORK 是临时逻辑库，在当前 SAS 会话结束时系统会自动删除。如果用户想要在服务器磁盘上永久保存该临时导入的数据集 WORK.CLASS，可以在导入之前

指定一个非临时逻辑库来保存数据。用户也可以在 SAS 代码中新建一个非临时逻辑库 mylib，然后在 mylib 中生成一份 WORK.CLASS 的复制。

```
libname mylib "/home/yinliangwu0";
data mylib.class;
  set work.class;
run;
```

这样用户就可以将打印 class 数据集的代码改为（见图 1-25）：

```
proc print data=mylib.class;
run;
```



图 1-25　在服务器上生成永久性数据集

运行上面的代码后，用户可以看到服务器的“逻辑库 / 我的逻辑库”多了 MYLIB 逻辑库，其中包括数据集 CLASS。即使用户关闭 SAS Studio，这个数据集也会一直存储在服务器磁盘上。这样当用户下一次登录 SAS Studio 时可以继续使用该数据集（见图 1-26）。

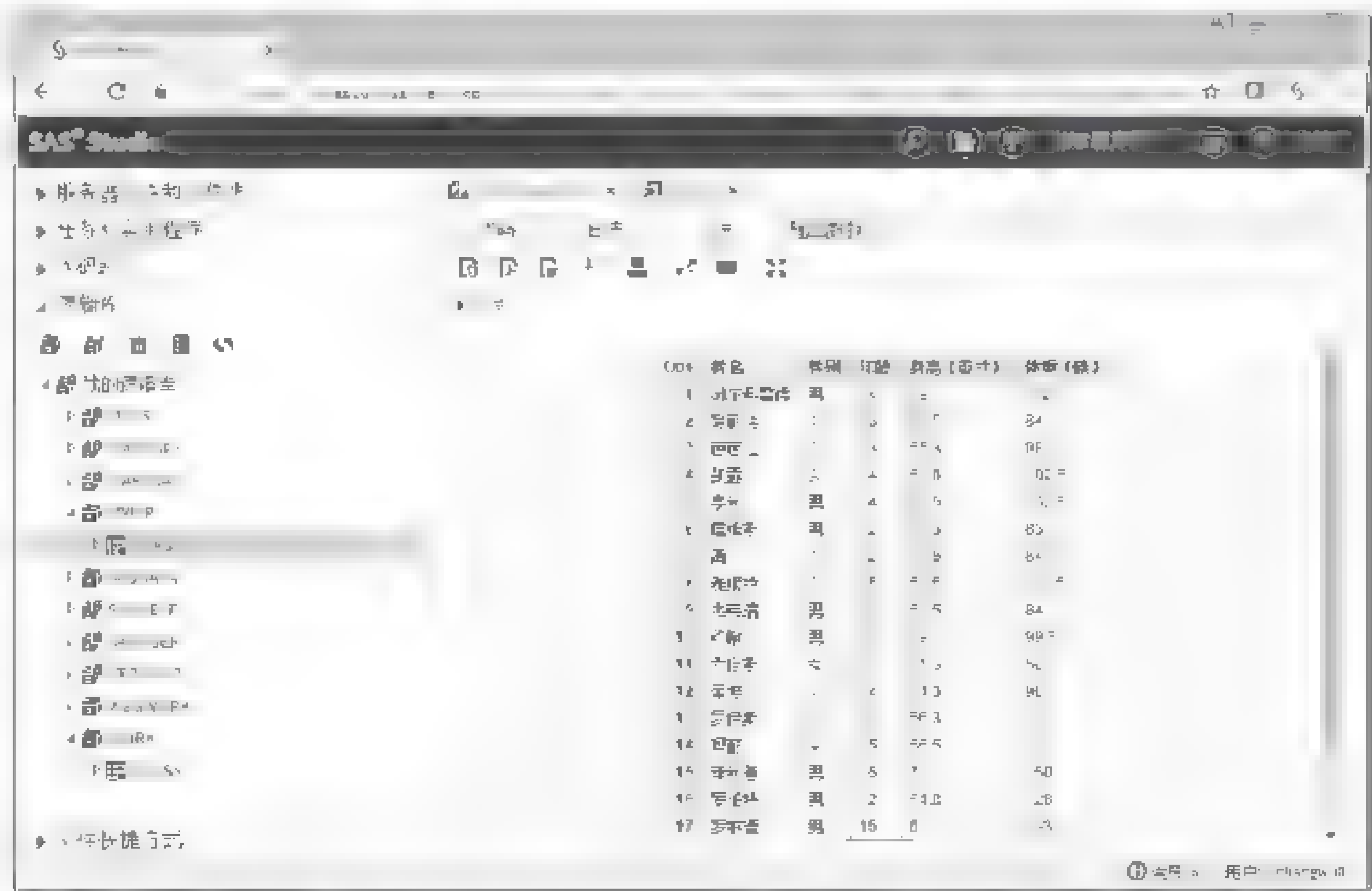


图 1-26　永久性逻辑库 MYLIB 中的数据

该数据集对应的物理文件位置就是远程服务器 odaomr.oda.sas.com 的磁盘路径 /home/yinliangwu0/class.sas7bdat（见图 1-27）。

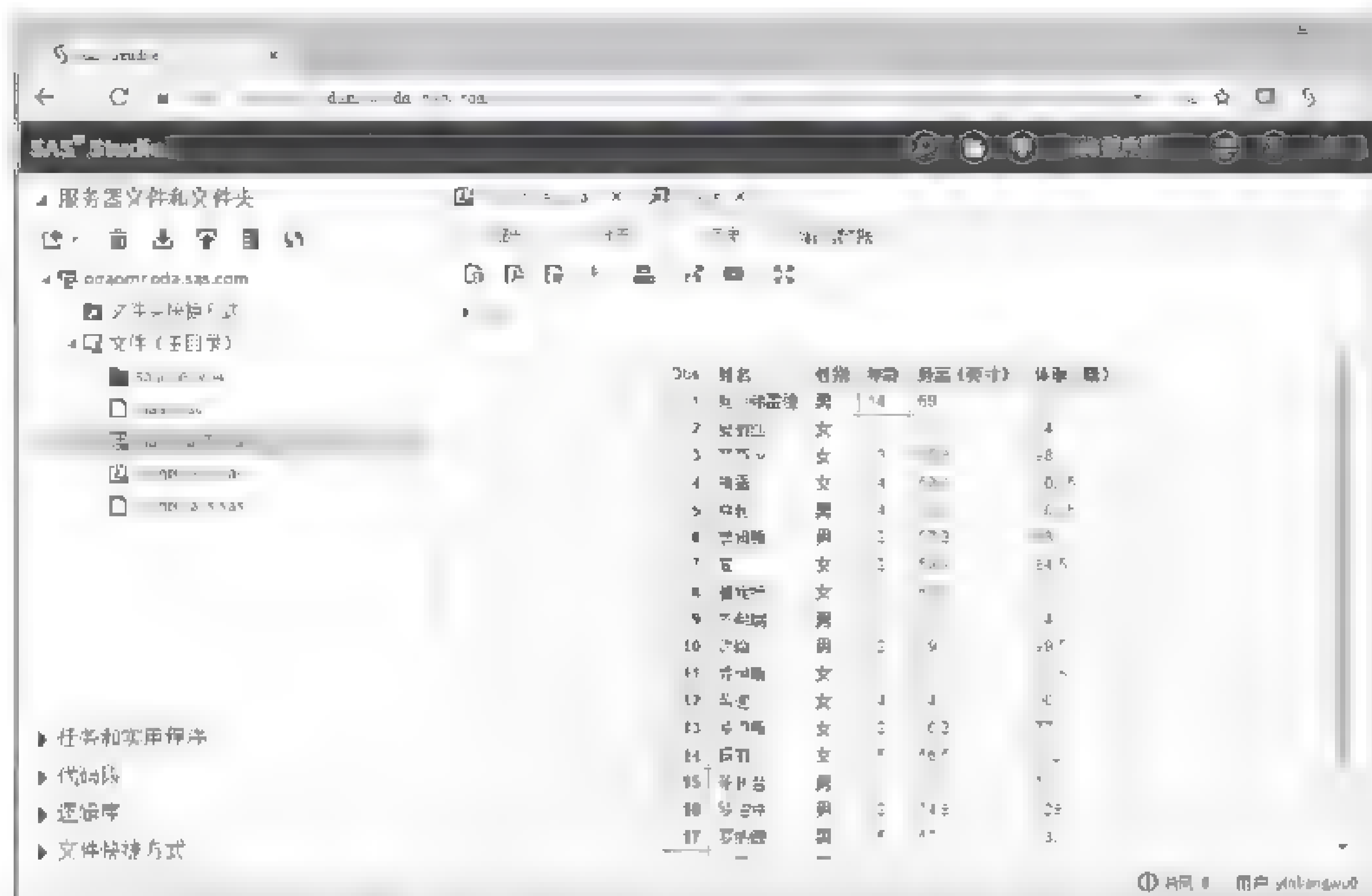


图 1-27　永久性逻辑库路径的数据文件

至此，用户就完成了数据集的上传，并将数据保存在非临时逻辑库 mylib 。同时也在服务器上也编写了 SAS 程序 /home/yinliangwu0/PrintClass.sas。这样我们就可以随时退出 SAS Studio 环境。退出 SAS Studio 应用只需点击右上角的“注销”菜单即可(见图 1-28)。

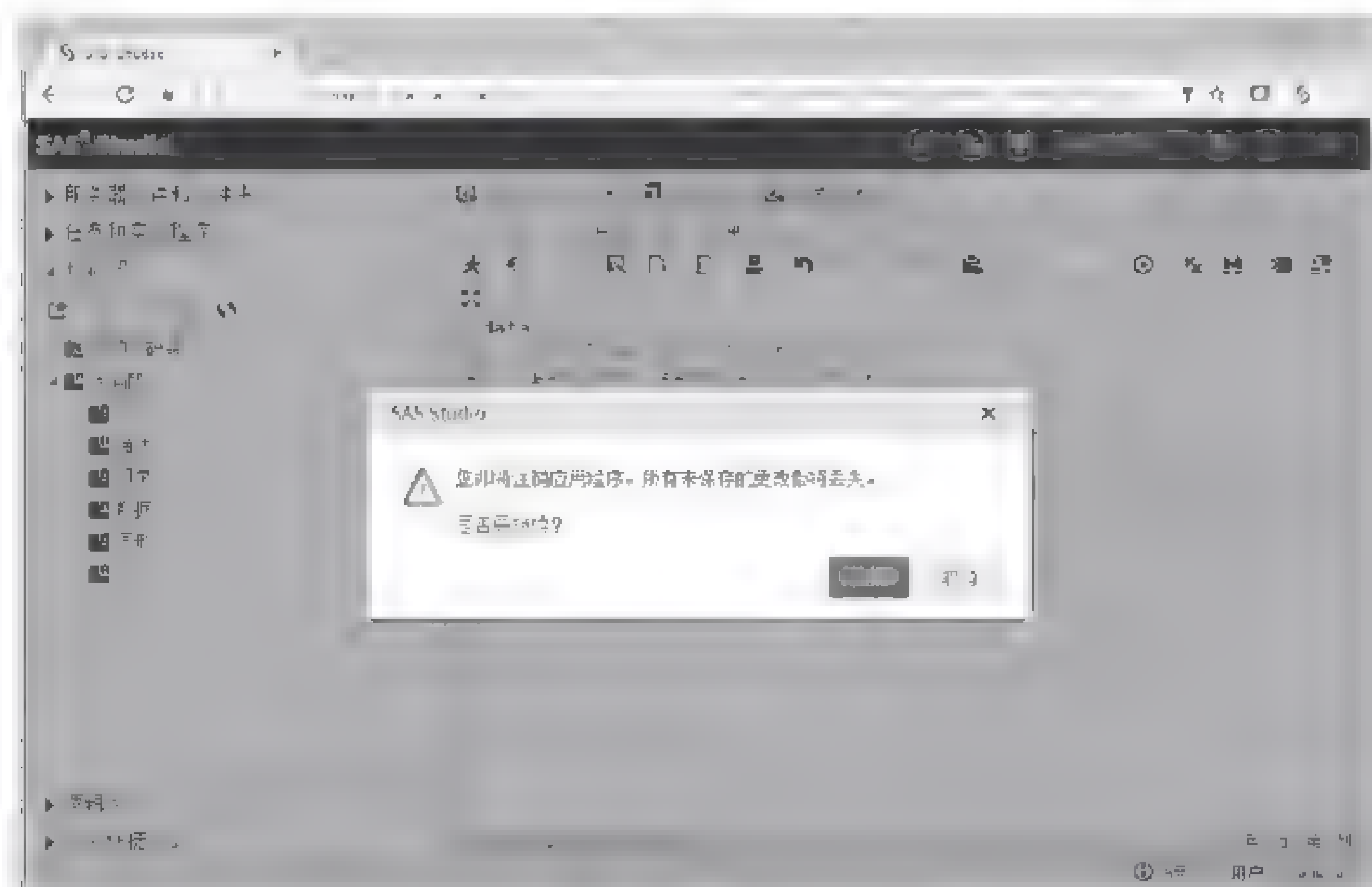


图 1-28　注销 SAS Studio

安全退出后显示如下页面，应关闭浏览器清除客户端数据以提高安全性（见图 1-29）。



图 1-29　已安全注销窗口

由于所有数据都保存在云端，因此用户下次登录时所有的数据和程序都还在（见图 1-30），用户可以随时继续自己的分析工作，也可以随时保存工作并退出。

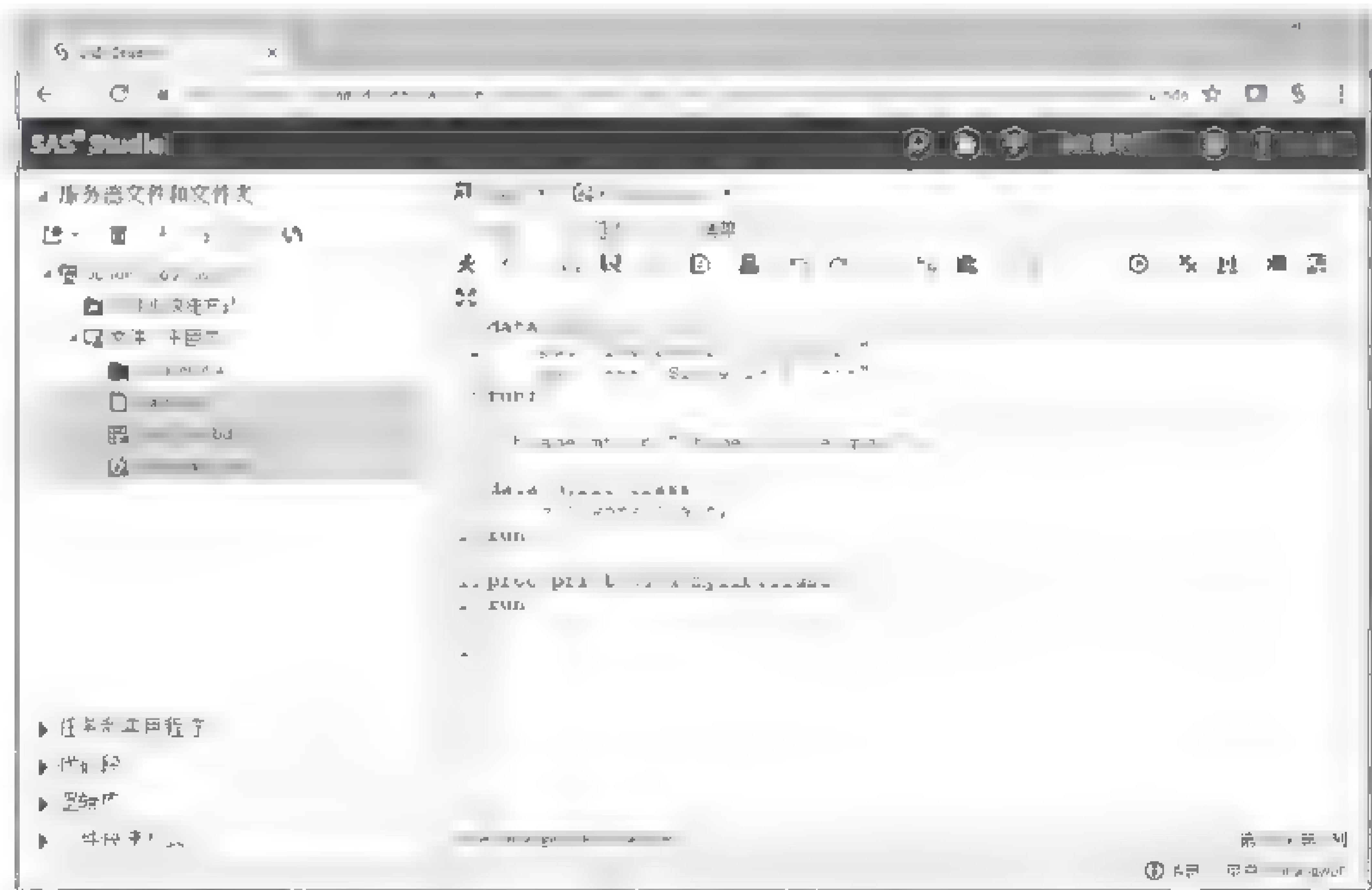


图 1-30　服务器文件和文件夹

关于云端的 SAS Studio 环境，以下重要提示信息可用于检查版本或者设置系统选项。

（1）检查 SAS Studio 和后台 SAS 系统的版本信息：点击右上角“?”号，然后选择“关于 SAS® Studio”即可查看（见图 1-31）。

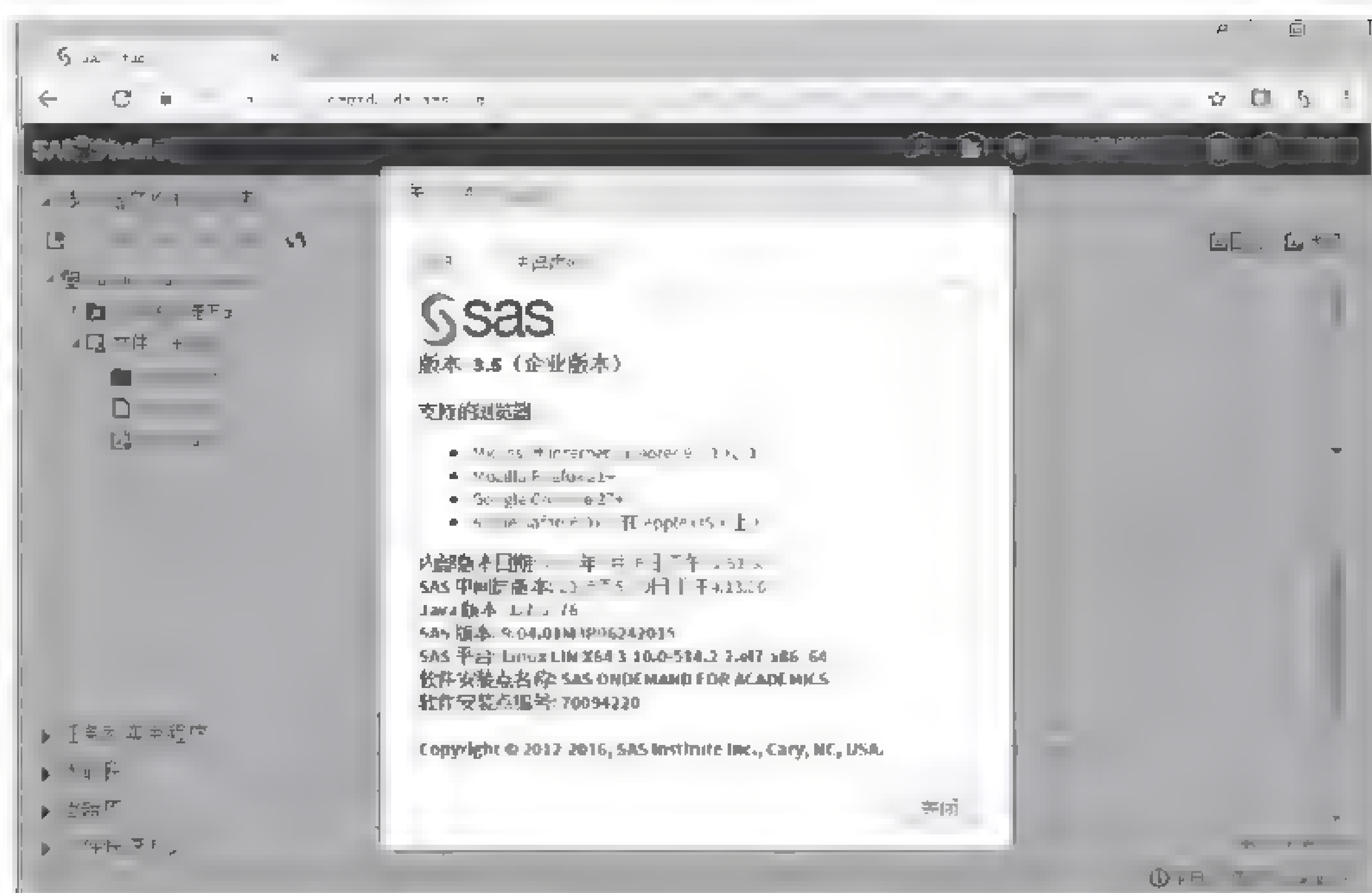


图 1-31　检查 SAS Studio 和后台 SAS 版本号

（2）查看系统默认的文本编码：单击右上角“其他应用程序选项”按钮，选择“参数选择→常规→ 默认文件编码：UTF-8”，该选项设定读取和保存 SAS 程序代码的默认编码格式（见图 1-32）。

在中文平台上创建的 SAS 程序文件默认是 GB2312 编码。而 SAS SODA 系统默认采用 UTF-8 处理文件读写和数据，因此建议在上传文件 / 上传代码之前确保文件为 UTF-8 编码。该选项并不影响前面提到的数据导入功能，因此即使修改此文本编码为 GB2312，也不会修正从客户端上传 GB2312 编码的数据文件导入 SAS 系统环境，更不会有文件编码 GB2312 和系统默认会话编码 UTF-8 不一致而导致的乱码问题。

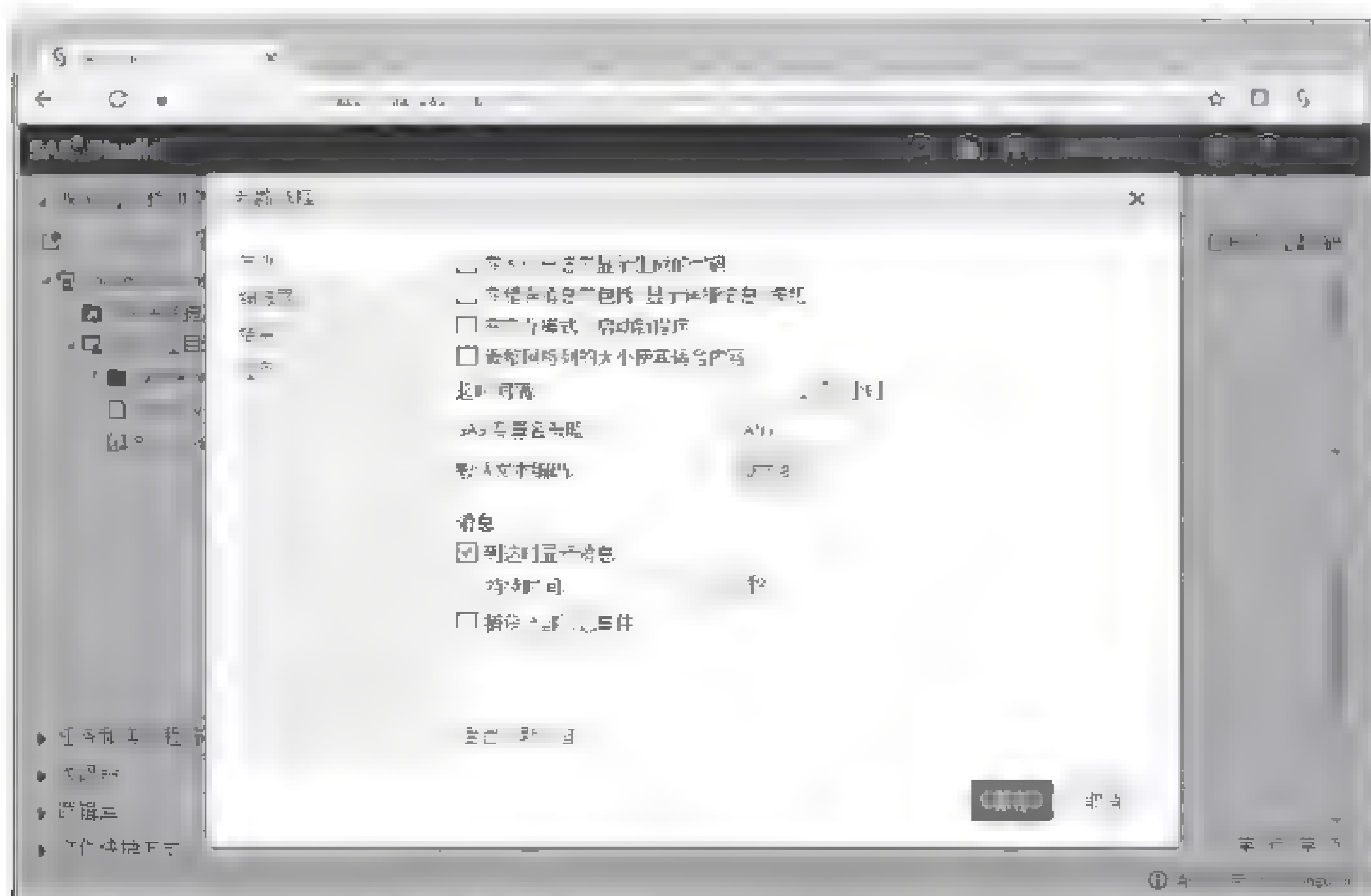


图 1-32　SAS Studio 参数选择窗口（常规）

（3）在参数选择窗口“启用提示”可以改善程序编辑器的行为，从而让用户在进行 SAS 编程的时候获得必要的上下文提示信息（见图 1-33）。



图 1-33　SAS Studio 参数选择窗口（编辑器）

启用选项后 SAS Studio 编辑器能动态提示帮助信息（见图 1-34）。

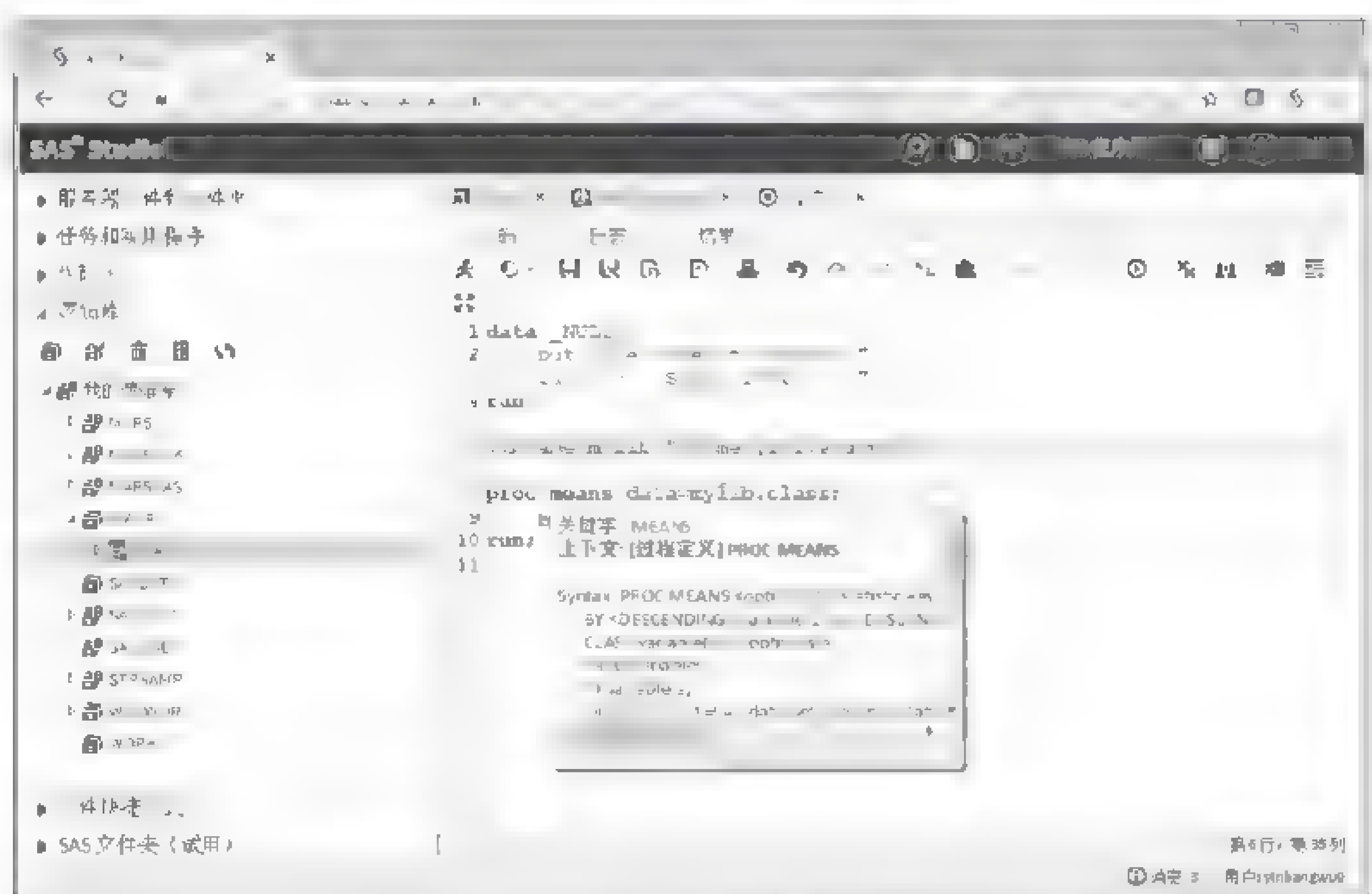


图 1-34　SAS Studio 动态提示信息

（4）在 SAS Studio 中利用“代码段”可以方便让用户重用某些代码片段。比如我们可以在代码编辑窗口中任意选中某些代码（见图 1-35），然后单击工具栏“添加至我的代码段”按钮，即可命名特定代码片段。

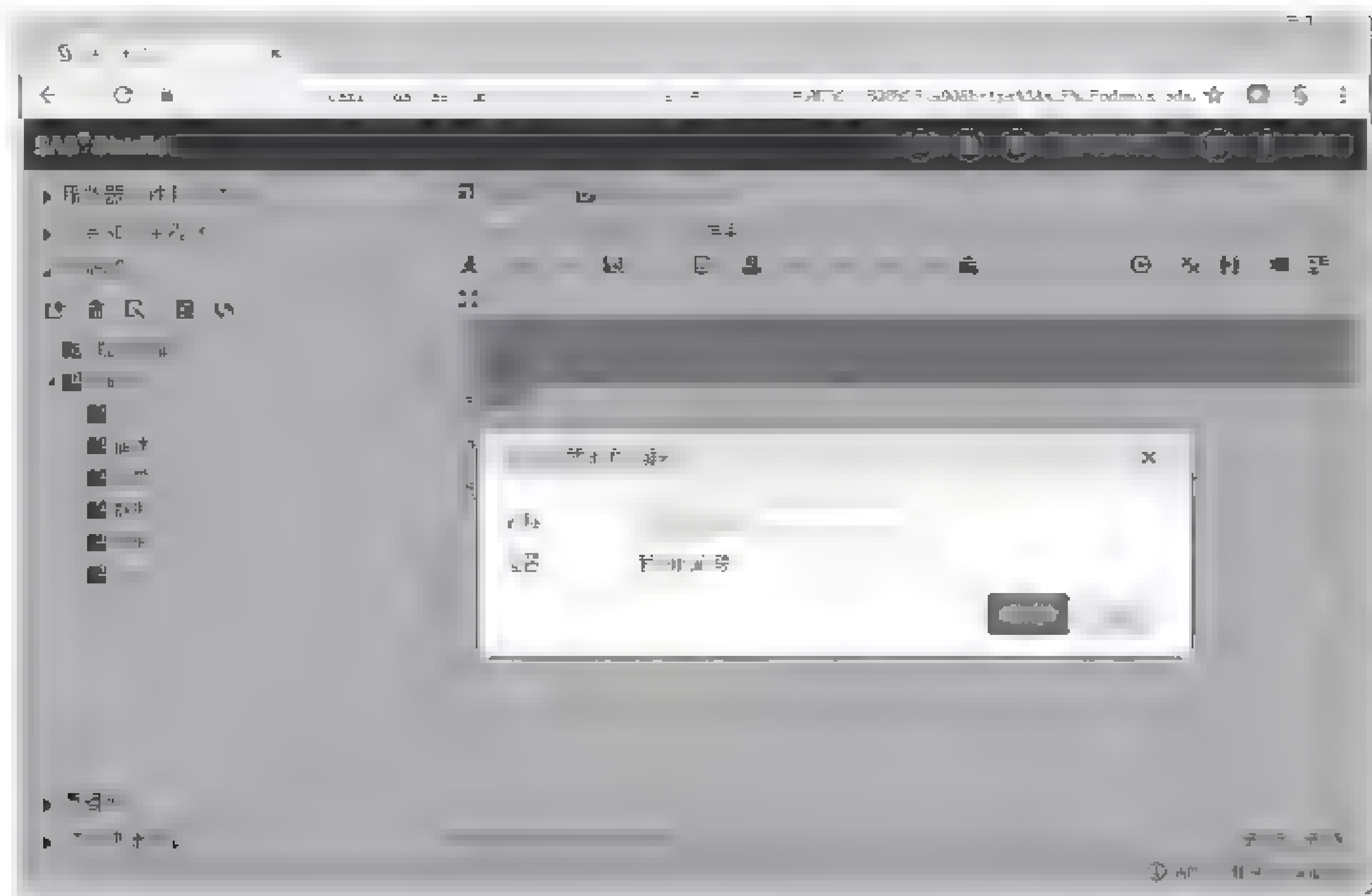


图 1-35　SAS Studio 添加代码片段

这样用户在新建另一个 SAS 程序时就可以很方便地在程序编辑器中通过双击左侧的代码片段（见图 1-36）将它添加到当前光标处。

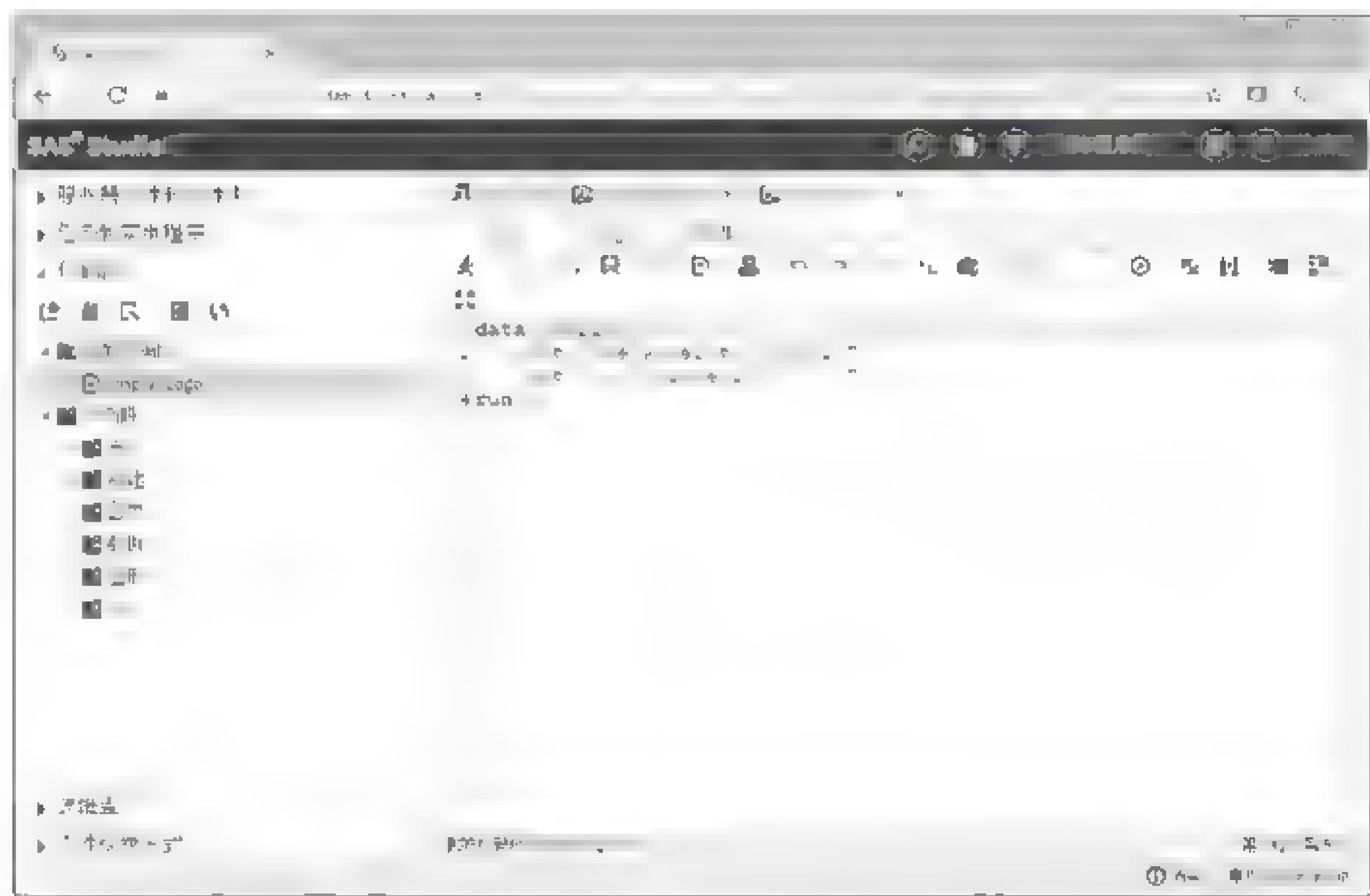


图 1-36　SAS Studio 我的代码段

比如下面编写的一个简单的显示 MYLIB.CLASS 平均值等简单统计量信息的程序（见程序 1-11）。

程序1-11　计算 MYLIB.CLASS 的均值（假定列名已改为中文）

```
libname mylib "/home/yinliangwu0";
proc means data=mylib.class;
  var "年龄"n "身高(英寸)"n "体重(磅)"n ;
run;
```

运行我们的代码然后保存到 MeansClass.sas。当我们需要插入 DisplayLogo 信息时，只需要将光标定位到特定位置，然后双击左侧的代码段即可实现自动代码插入（见图 1-37）。

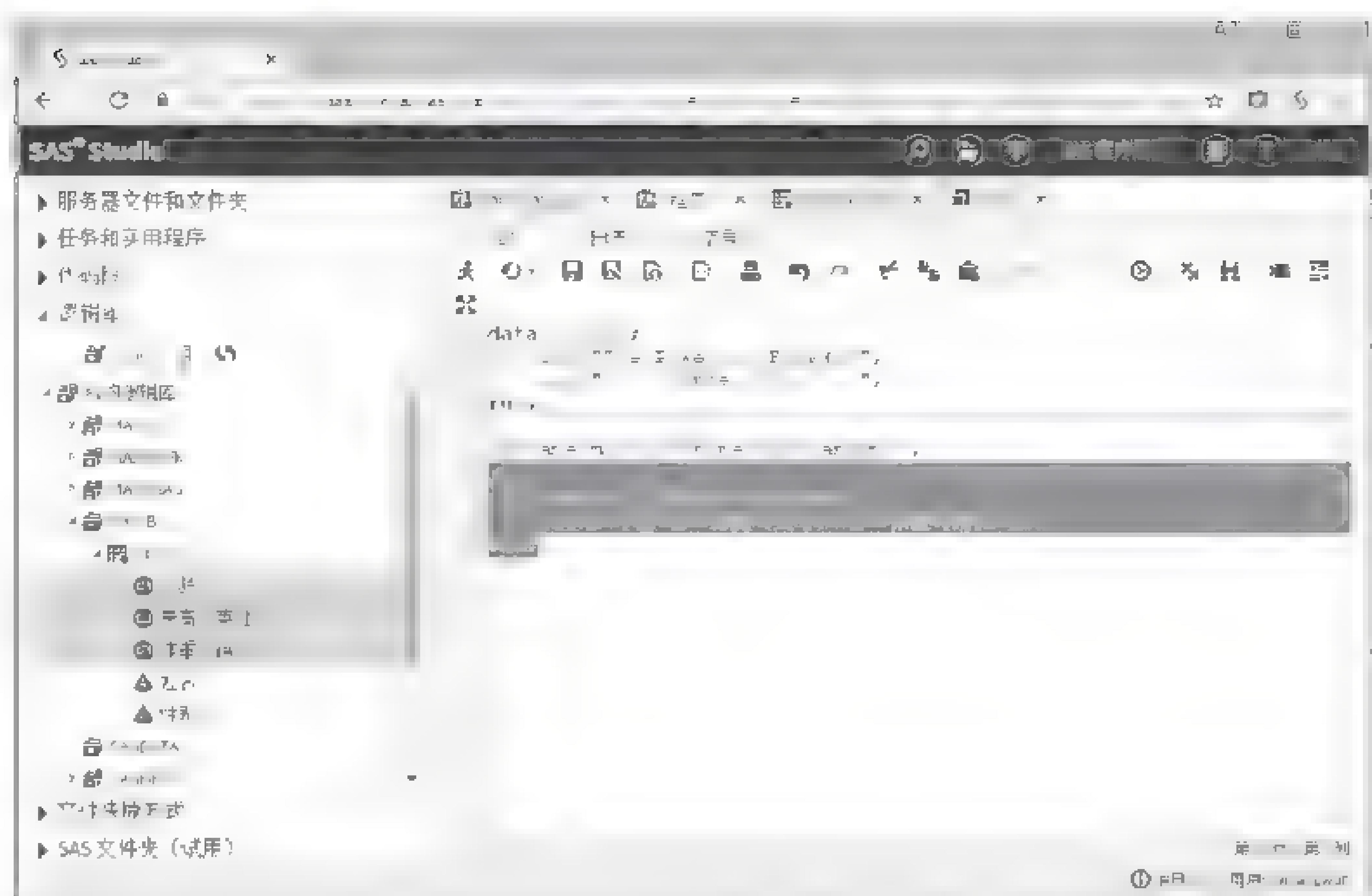


图 1-37　SAS Studio 插入代码片段

（5）创建自动执行代码：SAS 支持在系统启动时自动执行某些程序。比如我们希望保存在磁盘特定目录中的数据文件，在每次登录 SAS 工作环境时都可以用于分析，此时我们可将 libname 语句指向它所在目录，并加入 Autoexec.sas 文件中即可（见图 1-38）。

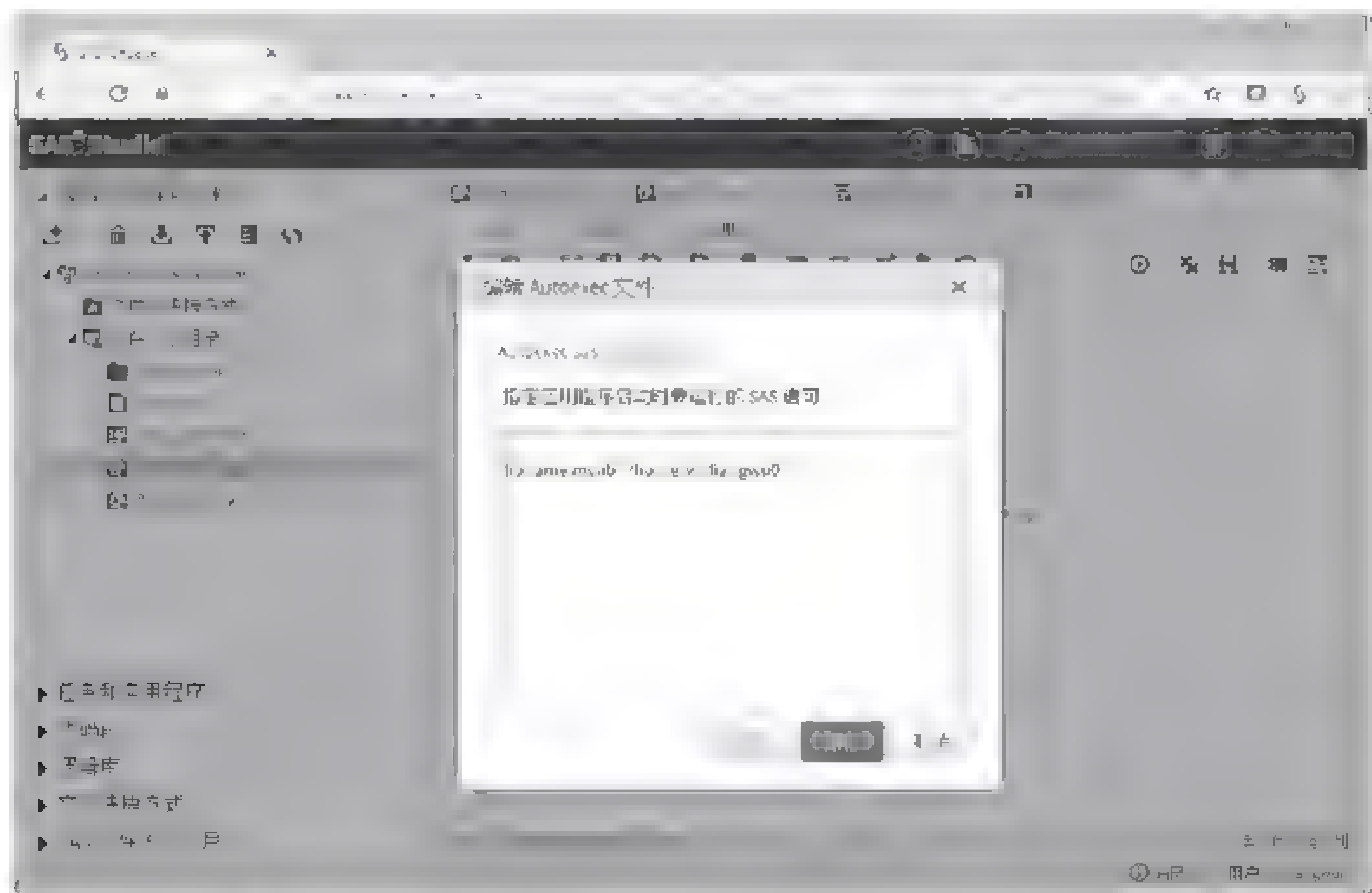


图 1-38　SAS Studio 自动执行代码（Autoexec.sas）

相应的原代码中 libname mylib “/home/yinliangwu0”; 语句就可注释掉或删除。SAS 中的 autoexec.sas 文件是 SAS 会话启动时系统自动执行的代码段。对于 SODA 上的 SAS Studio 用户，该 sas 文件在用户每次登录 SAS Studio 时都会自动执行。因此，/home /yinliangwu0 路径中的数据集都会默认出现在 mylib 逻辑库中，供后续分析处理用。

第 2 章

数据集与 DATA 步

操纵数据是数据学家的重要工作内容之一，主要用来为数据分析或创建报表准备必要的内容。传统商业智能包括数据仓库建模（Data Warehouse Modeling）、联机分析处理（Online Analytical Processing, OLAP）和数据挖掘（Data Mining, DM）。其中数据仓库建模后的数据准备工作由 ETL 或 ETCL 作业完成，具体包括数据的抽取（Extract）、转换（Transformation）、清洗（Clean）和加载（Load）等内容，传统的 ETCL 在现代商业分析领域已发展为包括各种数据转换处理的交互式数据管理（Data Management）系统，数据处理也是数据科学家除了数据分析之外的日常工作。

SAS 组织管理数据的最基本单位是 SAS 逻辑库（SAS Library）和 SAS 数据集（SAS Dataset）。细心的读者也许会记得 SAS 的 HelloWorld 程序，第一行都是以 Data 语句开头。那是因为 SAS 语言就是面向数据分析而设计的专门语言，在 SAS 的程序世界里数据是分析的基础，它是数据到信息、知识到智能整个分析链条的基石。

2.1 SAS 逻辑库

SAS 逻辑库是 SAS 面向数据处理而设计的存储和引用单位，是 SAS 组织数据的顶级单位。一个 SAS 逻辑库可以包含若干成员（Member），其中最常用的成员为 SAS 数据集（SAS Dataset）。SAS 逻辑库和 SAS 数据集的概念可分别对应传统关系型数据库（RDBMS）的数据库和数据表这两个概念，但 SAS 逻辑库比数据库包含更丰富的数据内容和更灵活的结构。

SAS 系统预定义了若干系统逻辑库，每次启动 SAS 运行环境（也称为建立一个 SAS 会话）这些系统逻辑库就已经自动为用户建立。根据逻辑库的内容在 SAS 会话结束后是否存在，可以将逻辑库分为永久库和临时库。一般情况下 SAS 默认包含 5 个永久库和 1 个临时库 Work，其中 5 个永久逻辑库分别为 Sashelp、Sasuser 以及 3 个地图专用逻辑库 Maps、Mapssas 和 Mapsgfk。（如图 2-1 左侧“SAS 资源管理器”所示）。

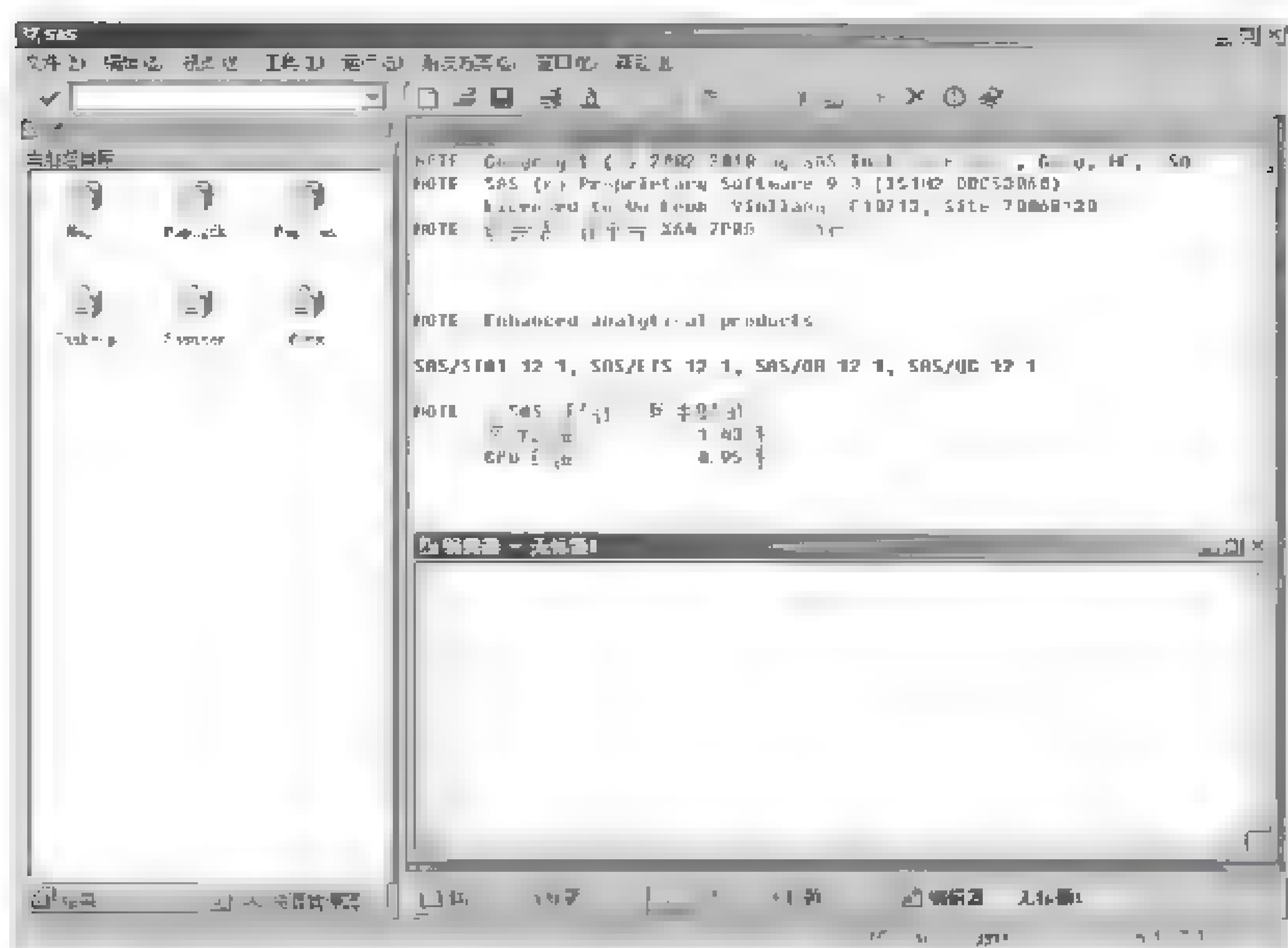


图 2-1　SAS 系统逻辑库

（1）临时库：在每次启动 SAS 运行环境的时候，SAS 都会建立一个临时逻辑库 Work。临时库 Work 用于在 SAS 会话期间临时存储和访问数据，当 SAS 会话结束，即退出 SAS 运行环境后，临时库 Work 和它的内容会被 SAS 系统自动删除。

在 Windows 平台的 SAS 环境中，每次启动一个 SAS 会话，SAS 系统会在操作系统临时目录（系统环境变量%TEMP%所指定）下的“SAS Temporary Files”目录中创建一个会话特定的临时路径（比如：C:\Users\sbjyiw\AppData\Local\Temp\SAS Temporary Files_TD8212_SBJYIW_），这是 SAS 当前会话存活期间存放各种临时数据的磁盘路径。

（2）永久库：永久库中的数据并不因 SAS 会话结束而消失，也就是 SAS 运行结束后存在于永久库的那些数据依然存在于磁盘或数据库服务器。SAS 会话中的永久库由 SAS 启动过程中使用的那个配置文件 sasv9.cfg 指定。

由于 SAS 支持几乎所有的语言和编码，在 SAS 安装环境中 sasv9.cfg 配置文件有多个，默认提供英文版、支持 DBCS 的英文版、Unicode 版和其他十余种语言特定的版本。它们分别对应 SAS 安装后在 Windows 开始菜单中的各启动项。sasv9.cfg 也支持使用 -config 参数指向另一个配置文件 sasv9.cfg 来重定向系统设置，从而构建多层次的配置文件体系。

Sashelp 逻辑库：它是系统预定义用于提供系统初始化后可用数据的系统逻辑库。逻辑库 Sashelp 在 sasv9.cfg 中由以下配置被映射到多个目录，表示从这些目录中读取可用的数据集文件，并在 SAS 系统中随时可用。这样多个目录中的文件在 SAS 会话中都可使用 SASHELP.* 的形式进行引用。其中 !SASROOT 和 !SASCFG 分别是磁盘上 SAS 的根路径和配置文件路径。

```
-SASHELP (
        "!SASCFG\SASCFG"
        "!SASROOT\nls\zh\sashelp"
        "!SASROOT\core\sashelp"
        ...
)
```

Sasuser逻辑库：它是系统预定义用来在SAS会话运行期间存放当前用户特定的数据。一般用于隔离不同SAS用户的数据。比如在Windows系统上，多个用户使用安装在同一路径下的SAS软件，各用户对应的Sasuser逻辑库会被分别映射到不同的用户主目录。在以服务器模式运行的SAS环境中，来自客户端的多个用户在服务器上具有不同的映射目录。

逻辑库Sasuser在sasv9.cfg配置文件中是使用-SASUSER指定的，通常对应于操作系统中的用户主目录或者数据库中的特定用户数据库。比如在Windows系统中，-SASUSER被指向“?CSIDL_PERSONAL\My SAS Files\9.3”，对应的操作系统路径为用户目主目录下的某个地址，如C:\Users\sbjyiw\Documents\My SAS Files\9.3。

（3）用户自定义逻辑库：它是用户在SAS程序中用LIBNAME语句建立的若干用户库，用户用它来引用存放于磁盘或者数据库服务器的持久化数据。程序2-1基于系统数据集SASHELP.CLASS创建了用户数据mylib.foo，该存储实体保存在目录C:\temp中。如果希望用户自定义逻辑库在每次启动SAS时都可用，可将Libname语句放到自动执行的autoexec.sas文件中。

程序2-1 自定义逻辑库

```
libname mylib 'c:\temp'; *定义逻辑库mylib;

*在 mylib 中生成sashelp.class 的拷贝，存于 c:\temp\foo.sas7bdat;
data mylib.foo;
  set sashelp.class;
run;
*打印我的数据 mylib.foo，实际上引用 c:\temp\foo.sas7bdat;
proc print data=mylib.foo;
run;
```

如果数据存储是基于数据库而非文件系统，则SAS访问引擎将会隔离这种复杂性，只需要提供类似ODBC数据访问连接字符串即可。比如连接最常用的SQL Server和Oracle数据库可以使用如下类似代码。其中对于Oracle数据库还要求安装Oracle Database Instance Client客户端访问软件，其PATH键值可以直接写连接串，也可以是C:\Program Files (x86)\Oracle\Instant Client\network\admin\tnsnames.ora文件中的访问入口，如果XXX为如下第二段path=后面引号内的字符串，则其libname可以简化。

```
/* 连接 SQL Server 数据库：OLEDB 访问方式*/
Libname mylib oledb user=sa password=foo datasource="192.168.1.123"
  provider=sqloledb properties=("initial catalog"=MYDB);

/* 连接 Oracle 数据库：需 Oracle 数据库访问客户端*/
libname mylib oracle user=myuser password=mypasswd
  path="(DESCRIPTION=(ADDRESS = (PROTOCOL = TCP)(HOST =
  192.168.1.123)(PORT = 1521))(CONNECT_DATA = (SERVICE_NAME=ORCL)))" ;

libname mylib oracle user=myuser password=mypasswd path=XXX ;
/* 需在 tnsnames.ora 文件中指定访问入口 XXX=(...)，括号中内容为PATH 字符串值*/
```

不管SAS运行在Windows操作系统还是UNIX操作系统，你的SAS代码总是可用相同的逻辑库引用名（Library Reference Name）来引用你的数据。逻辑库引用名是当前

SAS 会话能够识别的逻辑名称，指向某个操作系统能够识别的物理位置或者数据访问引擎的访问入口。SAS 逻辑库引用的根本作用是在 SAS 代码中隔离了操作系统或数据库系统的物理位置，提高了 SAS 代码的可移植性，也就是说任何数据在 SAS 代码中引用的时候都是“逻辑库 . 数据集”，这样就保持了 SAS 分析代码的一致性和简洁性。

用户在任何需要时可改变某个逻辑库引用的物理指向或者清除已经分配的某个逻辑库引用，比如程序 2-2 改变逻辑库 mylib 指向：

程序2-2　改变逻辑库指向或清除逻辑库

```
libname mylib "C:\windows";
*注意：改变逻辑库 mylib 的指向为 C:\windows;

libname mylib clear;
*注意 清除逻辑库 mylib,此语句后 mylib 将不再可用，SAS并不删除C:\temp 目录中任何物理文件，
只切断了SAS中的引用标识而已;
```

在 SAS 代码里，如果你想要删除 C:\temp 目录中的物理文件 foo.sas7bdat, 可以调用如下语句 (见程序 2-3)：

程序2-3　删除特定逻辑库中的数据集

```
proc datasets library=mylib;*磁盘上的物理文件也相应删除;
  delete foo;
run;
```

在 Base SAS 运行环境中，你可以用鼠标右键单击逻辑库来查看逻辑库的属性，可查看到它所映射的磁盘路径（见图 2-2）。

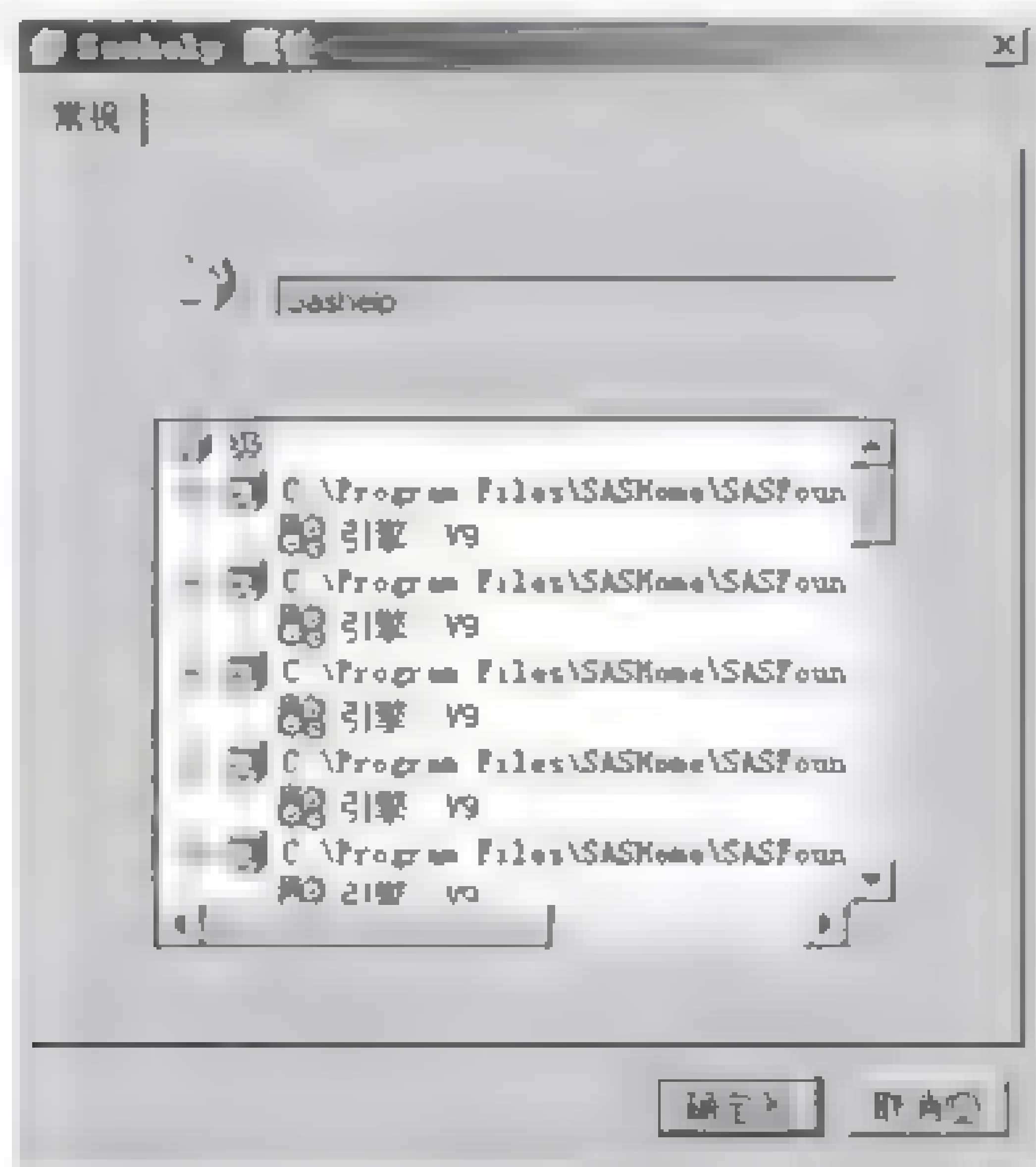


图 2-2　SASHELP 逻辑库属性

用户也可用如下代码查看 Sashelp 映射的目录和包含的内容摘要，相当于列出数据库的摘要信息（见程序 2-4）。

程序2-4　列出逻辑库信息

```
proc datasets library sashelp;
run;
quit;
```

你也可以使用 LIBNAME 语句的 LIST 功能在日志窗口中显示特定逻辑库的粗略信息，也可使用关键字 _ALL_ 列出系统中所有可用逻辑库的映射信息（见图 2-3）。

```
libname mylib list;
```

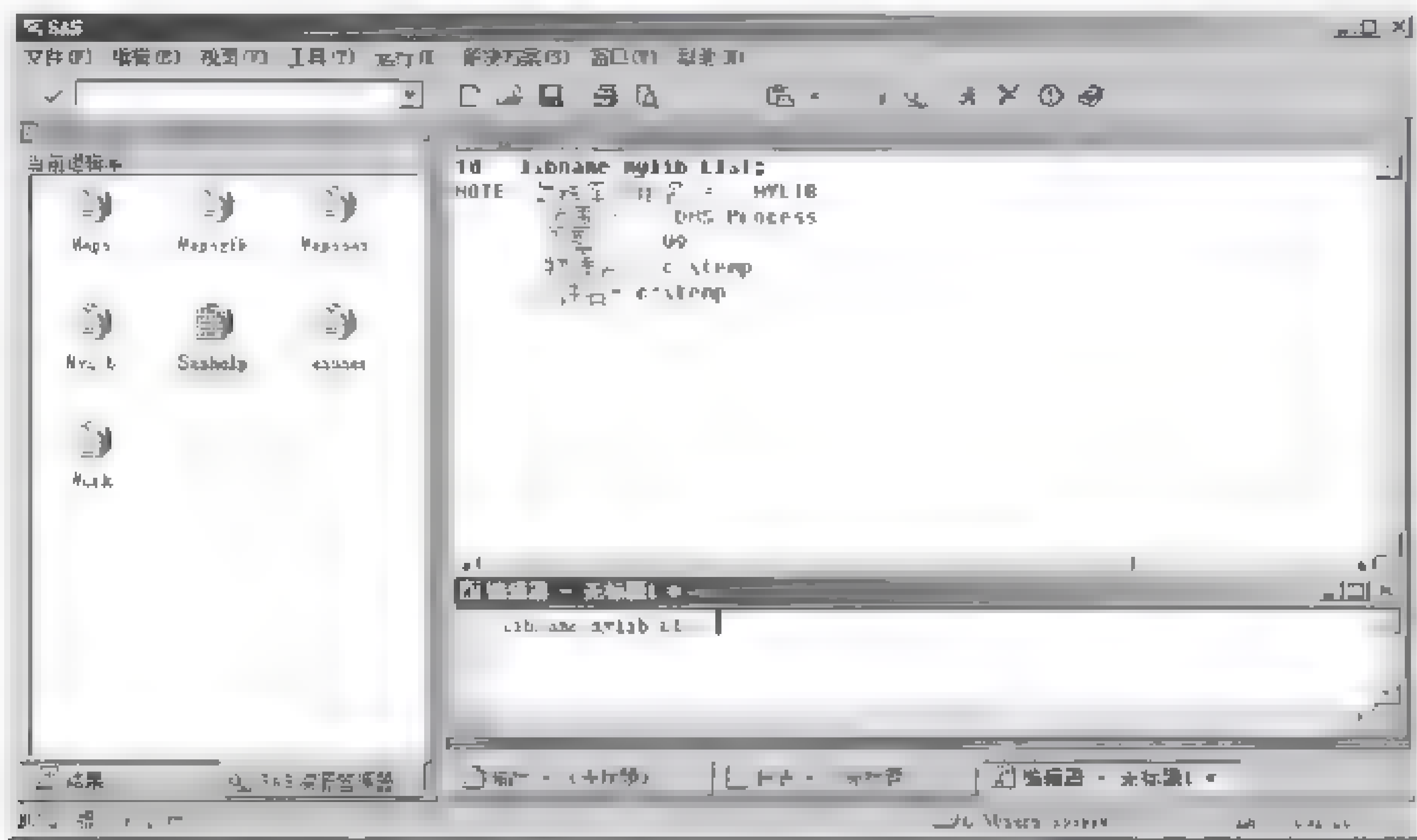


图 2-3 代码中列出逻辑库信息

```
libname _ALL_ list;
```

在日志窗口中将输出如下内容（见图 2-4）。

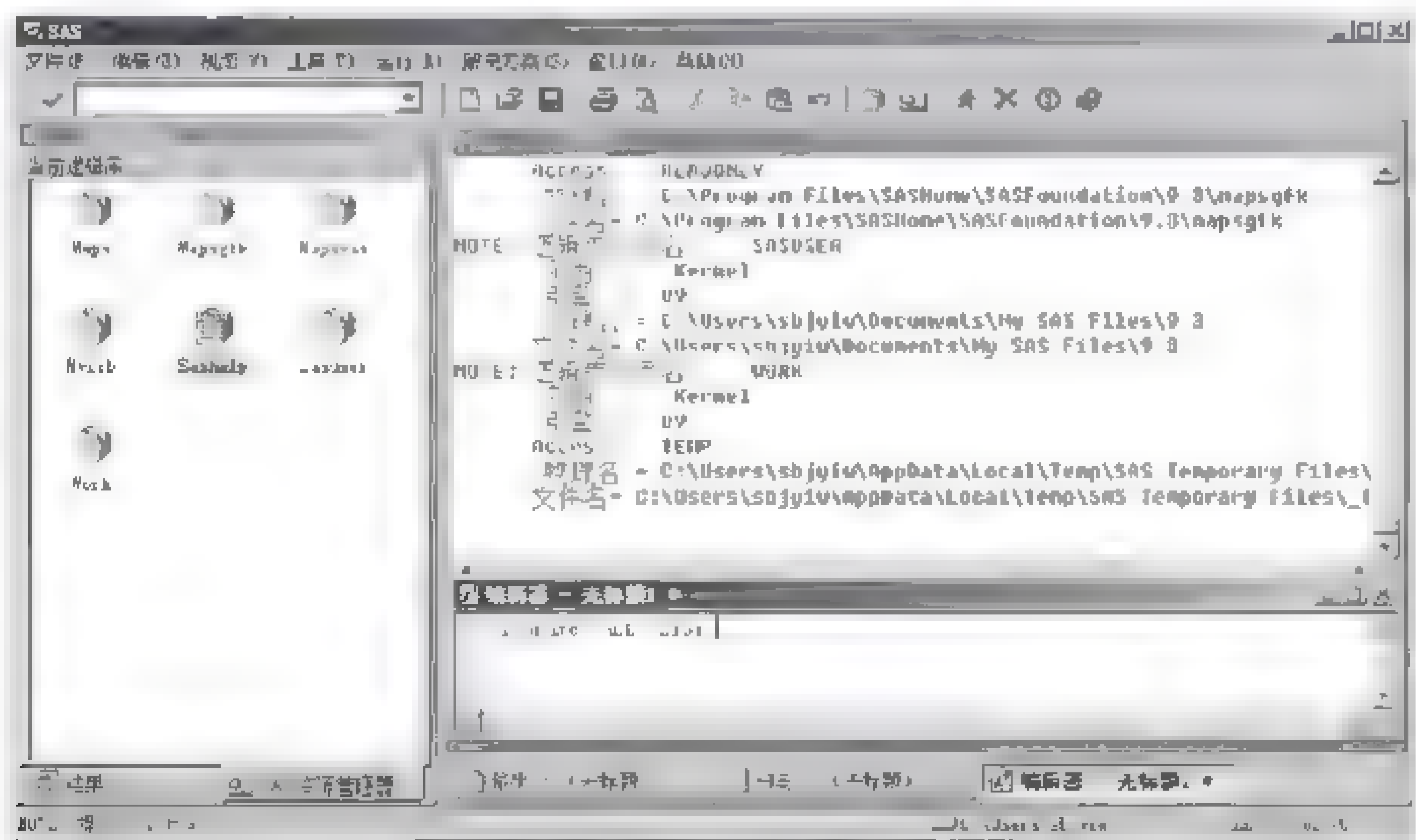


图 2-4 查看所有逻辑库信息

2.2 SAS 数据集

SAS 数据集是一种 SAS 特定的结构化数据文件，这种表状数据由变量（Variable）和观测（Observation）组成，变量和观测分别对应传统数据库中表的列和行。实际上，变量和观测这两个术语来自统计分析学科，表示对某个事件的属性或特征的连续观测结果。

SAS 数据集从结构上看包括描述部分和数据部分。描述部分是数据集的元数据，主要包括数据集的属性信息，包括数据集名称、文件编码、创建日期，观测数目（行数）、变量数目（列数）以及每一个变量的具体属性定义（包括名称、类型、长度、标签、输入/输出格式）等。数据部分就是每一行记录本身。要查看数据集的元数据（见图 2-5），可以使用 PROC CONTENTS 过程步来显示（见程序 2-5）。

程序2-5　查看数据集的元数据信息

```
proc contents data=sashelp.class;
run;
```

图 2-5　查看数据集的元数据

在 SAS 代码中，数据集使用两级名称 [LIBREF.]DSNAME 进行引用，如果是临时库 Work 中的数据，用户可以省略逻辑库引用名 LIBREF，也就说当未指定逻辑库名称时，系统默认使用临时库 Work 中的数据。

- SAS 数据集中的每一列，在SAS中称为变量。变量名称由字母、下划线和数字组成，名称必须以字母或下划线（不能以数字开始）开始。变量名称的长度范围为1~32字节。SAS内部使用大写形式创建变量，因此SAS代码中对变量名称不区分大小写；另外，SAS 步中语句需要指定变量列表时，除了可使用减号指定区间外，还可使用冒号 : 作为通配符，比如 var x2-x3; 或 keep x:; 等。
- SAS变量类型包括字符型和数值型，缺省都是8字节长。字符类型的变量值可为1~32767个字节长度的文本，数值型可为2~8任意字节长的整数或浮点数。其中在IBM大型机上运行的SAS支持 2~8字节长，而在Windows/Linux操作系统上运行的SAS支持3~8字节长。

在此谈论的 SAS 变量类型是传统意义上 SAS 数据集中的数据类型，即 DATA 步中的变量类型。在本书后面的章节中我们将看到 PROC DS2 能支持更加丰富的数据类型，但那些数据类型只是在运行时在内存中起作用，一旦将内存数据持久化输出到 SAS 数据集，则只有定长字符型和数值型两大类数据类型。程序员可能觉得数据类型偏少，但在统计科学中，数据所包含的信息由定类、定序、定距和定比四种变量描述，即变量只包

含类别变量（定类、定序）和量化变量（定距、定比）两大类，因此字符型和数值型完全可以覆盖所有的数据类型。相反地，传统编程语言中的字节型、短整型、长整型以及单精度，双精度浮点型更多的是面向计算机存储而设计的数据类型，并不是面向数据分析关注变量本身所包含的信息量而设计的数据类型。

- SAS变量的输出格式用于指定该变量（列）的默认输出格式 这样一来相同的SAS内部存储数据可以根据不同的输出格式生成不同的字符串表达 SAS 变量的输入格式用于告诉SAS按照该输入格式从外部数据源读取数据到SAS 数据集中。后面一节将详细讲解SAS 的输入格式和输出格式。

SAS输入/输出格式

SAS 输入 / 输出格式在表现形式上就是一个包含小数点“.”的特殊文本标记，它不需要用引号括起来。完整的 SAS 格式定义包括可选的格式前缀 $（如果是字符型格式的话）、格式名称（FormatName）、输出宽度 w 以及数值型变量特有的小数点位数 d 等信息。SAS 输入 / 输出格式完整语法形式如下：

```
<$>格式名称<w>.<d>
```

SAS 输入 / 输出格式是数据的存储表达和 SAS 内部存储之间的转换桥梁。根据 SAS 数据类型的不同可以将 SAS 格式主要分为字符型格式和数值型格式两大类，其中数值类型格式可细分为数值型、货币型、日期 / 时间 / 日期时间型 3 个子类的格式。格式用于从 SAS 外部输入 SAS 系统的转换称为输入格式（INFORMAT），相反用于输出 SAS 变量的称为输出格式（FORMAT）。

字符型格式必须在格式名前面加美元符 $（源自英文字符串 String 的首字母） 表示，w 表示输出的总宽度；而数值型变量还可指定小数点后的位数 d。下面列出一些最常用的格式例子：

- 字符型格式：$w. 表示 w 个字节宽度的字符
- 数值格式：w.d
  （1）COMMAw.d 表示用逗号做千位分隔符，总长度为 w，包含 d 位小数。
  （2）COMMAXw.d 表示用“.” 作千位分隔符，逗号作小数点，用于部分欧洲国家（如法国）的数值表示，功能与 COMMAw.d 类似。
- 货币格式：
  （1）DOLLARw.d 表示变量按美元格式输出：输出以美元符开头，逗号作为千位分隔符，使用标准小数点格式。
  （2）EUROw.d 表示变量按欧元格式输出：以欧元符号开头，逗号千位分隔符，标准小数点格式。
  （3）EUROXw.d 与EUROw.d 格式相同，但以“.”作千位分隔符，逗号作小数点符号。

运行程序 2-6 代码，读者可检查输出结果以了解 SAS 输出格式的作用：

程序2-6　SAS 字符型和数值型输出格式

```
data _null_;
  c="abcdefghijklmn"; *14bytes;
  put "char=" c;
  put "char=" c $;
  put "char=" c $14.;

  n=1234.567890;
  put "w.d=    " n 12.3;
  put " COMMAw.d=" n COMMA12.3;
  put "COMMAXw.d=" n COMMAX12.3;

  put "DOLLARw.d=" n DOLLAR12.3;
  put "   EUROw.d=" n EURO12.3;
  put " EUROXw.d=" n EUROX12.3;
run;
```

系统输出如下：

```
char=abcdefghijklmn
char=abcdefghijklmn
char=abcdefghijklmn
w.d=    1234.568
 COMMAw.d= 1,234.568
COMMAXw.d= 1.234,568
DOLLARw.d=  $1,234.568
  EUROw.d=  E1,234.568
 EUROXw.d=  E1.234,568
```

- 日期格式：

  （1）DATEw. 比如 DATE7. 显示英语国家的格式 16JAN17。

  （2）MMDDYYw. 比如 MMDDYY10. 显示 01/01/1960。

  （3）YEAR4. 显示年份信息，如 1960。

  （4）NLDATE. 显示本地语言格式的日期，此时 NL 开头的日期格式会根据不同的 SAS 会话 Locale 系统选项输出不同的结果，比如在中文 SAS 上显示 1960 年 01 月 01 日，在法文 SAS 环境中则按法文 Locale 的格式显示 01 janvier 1960。

- 时间：

  （1）TIMEw. 比如 TIME. 显示 22:16:27。

  （2）NLTIME. 显示本地语言格式的时间，如中文 SAS 上显示 22 时 16 分 27 秒。

- 日期时间：为前面日期和时间的组合

  （1）DATETIME. 比如 DATETIME. 显示 01JAN60:22:16:27。

  （2）NLDATM. 显示本地语言格式的日期时间，如 1960 年 01 月 01 日　22 时 16 分 27 秒。

程序 2-7 显示了 SAS 代码中日期、时间以及日期时间格式的基本使用。

程序2-7　日期和时间格式

```
data _null_;
  mydate=today();
  put mydate DATE.;
```

```
  put mydate MMDDYY.;
  put mydate YEAR.;

  mytime=TIME();
  put mytime TIME.;

  mydt=datetime();
  put mydt DATETIME.;

  put mydate NLDATE.;
  put mytime NLTIME.;
  put mydt NLDATM.;
run;
```

系统输出：

```
22JAN17
01/22/17
2017
17:18:31
22JAN17:17:18:31
2017年01月22日
17时18分30秒
2017年01月22日 17时18分30秒
```

在 SAS 中有一系列以 NL 字符开头的国家语言特定的输入 / 输出格式（如 NLDATE.），这些格式在不同的 SAS 会话中由于系统选项 locale 的设置不同，因此输出不同。比如在 zh_CN locale 下输出符合中国简体中文格式的信息，在法文 fr_FR locale 下则会按法文习惯的格式输出信息。格式化只是纯粹显示数据，它并不转换数据。因此，同样的数值以法郎显示时并不会根据货币汇率进行转换显示（见程序 2-8）。

程序2-8 国家语言特定的输入/输出格式（NL Format）

```
options locale=zh_CN;
data _null_;
  day=date();
  put day nldate.;
run;

options locale=fr_FR;
data _null_;
  day=date();
  put day nldate.;
run;
```

系统输出如下所示：

```
2017年11月08日
08 novembre 2017
```

SAS 格式为我们在同一SAS 程序代码中输出符合不同国家语言和风俗习惯的数据表示成为可能，这是国际化软件公司在将软件产品走向全球化市场时必须提供的产品特性。SAS 提供了非常全面的国际化格式支持。

SAS 组织数据的基本容器是 SAS 数据集，更上一级为 SAS 逻辑库。SAS 数据集包含描述数据的元数据部分（即数据表的属性以及列定义信息）以及真正包含数据的观测行组成。不管数据是存在磁盘上，还是存在外部数据库或其他应用系统中，在 SAS 程序

员的世界里都被统一成逻辑库引用进行数据访问。这样用户就根本不必关注数据存储的细节，而是将更多的精力放在数据分析和展现本身上。因此，不管是 Oracle 数据表还是 SQL Server 数据表，或是 SAS 数据集，在 SAS 程序里都是简单地通过“**逻辑库.数据集**”（如 myoracle.table2 或 mysqlsvr.table3）两级方式进行引用。

SAS 数据集中的观测一般有 3 种生成方式。

（1）通过 DATA 步内嵌数据行，或基于已有的 SAS 数据集或外部数据库系统中的表来生成。

（2）通过 PROC IMPORT 或者其他面向数据操作的 PROC 步，如 PROC SQL 来生成。

（3）通过面向分析的 PROC 步作为输出或副产品生成，一般是中间临时数据或最终分析结果数据集。

程序 2-9 生成了一个有代表性的 SAS 数据集（见图 2-6），可帮助理解 SAS 数据集的逻辑结构。

程序2-9 包含两种基本类型的数据集范例

```
libname mylib 'C:\temp';/*创建用户自定义逻辑库 mylib，指向 C:\temp*/

/*生成 mylib.mydata 数据集，设置数据集标签*/
data mylib.mydata (label="This is my first data");
  length column1 $32; /*定义一个字符型变量，32个字节长*/
  column1="The only constant is change";
  label column1="This is Char column";

  column2=1234.5678; /*定义一个数值型变量，默认8字节长*/
  format column2 dollar10.2;
  label column2="This is Num column";
run;

proc contents data=mylib.mydata; run;
proc print data=mylib.mydata; run;
```

按字母排序的变量和属性列表					
#	变量	类型	长度	输出格式	标签
1	column 1	字符	32		This is Char column
2	column 2	数值	8	DOLLAR10.2	This is Num column

Obs	column 1	column 2
1	The only constant is change	$1 234.57

图 2-6 简单数据集范例

2.3 DATA 步

在 SAS 程序中 DATA 语句用于开始一个数据步，后续为若干 DATA 步特定的 SAS

语句。SAS 数据步结束于下一个 DATA 步或 PROC 步开始之处，或者结束于后续显式指定的 RUN 语句。DATA 语句最常见的调用方式有如下几种。

（1）DATA 语句可不指定任何参数，则 DATA 步将自动在临时逻辑库 Work 中创建一个输出数据集 DATA#，其中 # 为从 1 开始不断增长的唯一整数。因此，每次执行代码都会在 Work 中创建新的数据集。程序 2-10 SAS 代码将在临时逻辑库 Work 中创建数据集 DATA1，包含 1 行 5 列数据。再次运行该代码则将生成 DATA2，依此类推（见图 2-7）。

程序2-10 变量输出到默认数据集

```
data;
  Name="Leon"; Sex="M"; Age=30; Weight=83.5; Height=175;
run;
```

	Name	Sex	Age	Weight	Height
1	Leon	M	30	83.5	175

图 2-7 Work.Data1 内容

这种方式一般用在生成临时数据集而且不关心输出数据集的名称时使用，由于每次执行都会在 Work 中生成一份新的数据，它对应在操作系统磁盘上一个单独的文件存储，因此一般不建议使用这种方法。通常情况下会显示指定输出数据集的名字会更好一些，避免反复执行数据分析任务将磁盘空间耗尽。

这种临时数据集的名称可用 SAS 系统宏变量 &SYSLAST 跟踪（见程序 2-11），该宏变量用于跟踪上一次生成的数据集名称，并在打印结束后将它从临时逻辑库中自动删除。

程序2-11 跟踪最近使用的数据集名称

```
data;
  Name="Leon"; Sex="M"; Age=30; Weight=83.5; Height=175;
run;
proc print;run;

proc datasets nolist ;
  delete %scan(&syslast,2);
quit;
```

（2）DATA 语句也可指定特殊名称 _NULL_，表示不输出任何目标数据集，此时数据步通常用于纯粹的计算，或出于调试 SAS 代码的目的，不希望生成默认数据集时通常使用此方法。程序 2-12 只在日志中打印最近使用的那个数据集的名称，并不生成任何数据集。

程序2-12 不输出到SAS数据集

```
data _null_ ;
  put "&SYSLAST";
run;
```

（3）DATA 语句也可指定多个输出数据集，从而实现在一个 DATA 步里输出多个

数据集。程序 2-13 可生成两份内容完全相同的数据集 data1 和 data2。

```
程序2-13　输出到多个数据集中
data data1 data2;
  Name="Leon"; Sex="M"; Age=30; Weight=83.5; Height=175;
run;
```

如果我们希望把 SASHELP.CLASS 中所有数据行分成 13 岁以下的一组和大于 13 岁的一组，形成两个数据集 classA 和 clasB，则可使用下面的代码简洁地实现（见程序 2-14）。

```
程序2-14　按年龄将数据集SASHELP.CLASS 分为两个数据集
data classA classB;
  set sashelp.class;
  if age<=13 then output classA; /*输出到数据集A*/
  else output classB;  /*输出到数据集B*/
run;
```

如果想把数据集纵向上进行分割，如在 classA 中保留 name, age 列，而在 classB 中保留 name weight, height 列，则可以使用如下灵活方式实现（见程序 2-15）。

```
程序2-15　将数据纵向分裂为两个数据集
data classA(keep=name age) classB(keep=name weight height);
  set sashelp.class;
run;
```

在数据分析实践中，SAS 数据集中的观测有各种各样的生成方式，主要包括下面几种。

2.3.1　内嵌数据行或外部数据文件

1. 利用内嵌数据行创建SAS数据集

一般情况下，如果我们要生成的分析数据比较小，且不希望有独立的外部数据文件，可以直接将数据本身嵌在 SAS 代码中，内嵌数据行又有如下几种不同方式。

（1）最常见的情况是我们有一系列的数据行，数据项之间用空格分隔。这种情况我们可以使用 DATALINES 语句配合 INPUT 语句直接生成。DATALINES 语句表示后面的行是数据行。由于数据行并非 SAS 语句，因此它不需要分号，但该文本是格式敏感的数据。另外，除了 DATALINES 语句外，我们也可使用 DATALINES 语句的别名 LINES 或 CARDS 语句，它们三者等价。程序 2-16 代码生成 2 个字符串变量和 3 个数值变量的 3 行数据。

```
程序2-16　利用内嵌数据行创建数据集
libname mylib "C:\temp";
data mylib.myclass;
  input Name $ Sex $ Age Height Weight; /*$ 表示字符型变量*/
  datalines;
Alfred M 14 112.5 69
Alice F 13 84 56.5
Barbara F 13 98 65.3
run;
proc print;run;
```

系统会生成如下数据集（见图2-8），存放在C:\temp目录中。与利用临时库Work不同，该数据集在你SAS会话结束后依然存在，存放在C:\temp\class.sas7bdat文件中。

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	112.5	69.0
2	Alice	F	13	84.0	56.5
3	Barbara	F	13	98.0	65.3

图2-8　由数据生成的MYLIB.CLASS数据集内容

磁盘上的数据文件名称依赖于操作系统的文件系统，如Windows平台的FAT/FAT32和NTFS文件系统，不区分大小写。而Linux/Unix平台则区分大小写，SAS则一律使用小写字母生成数据集文件名。如果我们要读取Unix/Linux平台上包含大写字母的文件名，需要在SAS会话中启用系统选项VALIDMEMNAME=EXTEND并且数据集的名称必须与磁盘上的文件名大小写完全匹配才能正常读取。

（2）如果数据行包含特定的分隔符，我们可以利用INFILE语句来指向系统特殊的文件引用DATALINES，并且使用语句参数DELIMITER='<分隔符>'来指定分隔符。指定分隔符时也可以使用十六进制方式，如DELIMITER='2C'X。它与下面的代码使用逗号分隔符等价（见程序2-17），代码输出结果与程序2-16相同。

程序2-17　指定分割符

```
data myclass;
  infile datalines delimiter=',';
  input Name $ Sex $ Age Height Weight;
  datalines;
Alfred,M,14,112.5,69
Alice,F,13,84,56.5
Barbara,F,13,98,65.3
run;
```

（3）如果数据行本身包含SAS语句的结束标志符分号“;”的话，则我们必须使用DATALINES4语句来标记后续数据行，该语句表示后续数据行是用4个连续的分号来标记数据行结束的。DATALINES4也可以使用该语句的别名为LINES4或CARDS4表示。程序2-18中字符型变量NAME的值包含分号，则我们必须以DATALINES4来输入数据。

程序2-18　数据包含分号

```
data myclass;
  input Name $ Sex $ Age Height Weight;
  datalines4;
Alfred; M 14 112.5 69
Alice; F 13 84 56.5
Barbara; F 13 98 65.3
;;;;
```

数据包含引号并且引号中的数据包含字段分隔符本身时，我们需要在INFILE语句上启用分隔符敏感数据选项DSD（DELIMITOR-SENSITIVE DATA，DSD）。此时

如果数据行中甚至还包含 SAS 语句结束符分号，则我们同时需要使用 DATALINES4 而非 DATALINES 语句进行标记。下面的代码详细展示了此时如何正确读取数据（见程序 2-19）。

```
程序2-19　数据包含分隔符和引号
data myclass;
  infile datalines4 dsd; /* DSD 使用逗号分隔，且引号文本中包括逗号*/
  input Name: $9. Sex $ Age Height Weight  Address: $32. Description $24.;
  datalines4;
Alfred,M,14,112.5,69, "Alfred's home Beijing, China", Alfred's
description;
Alice,F,13,84,56.5, "Alice's home, Beijing, China", Alice's
description;
Barbara,F,13,98,65.3, "Babara's adress, Beijing, China", Barbara's
description;
;;;;
```

读取后的数据中可包括空格、逗号与分号（见图 2-9）。

Obs	Name	Sex	Age	Height	Weight	Address	Description
1	Alfred	M	14	112.5	69.0	Alfred's home Beijing, China	Alfred's description;
2	Allice	F	13	84.0	56.5	Allice's home Beijing, China	Allice's description;
3	Barbara	F	13	98.0	65.3	Barbara's adress, Beijing, China	Barbara's description;

图 2-9　读取数据中包含空格和分号

（4）如果数据行本身变长，也就是说数据行参差不齐，其中字符型变量又包含空格字符。此时我们需要使用 SAS 提供的列指针来明确指定每一个数据行中变量读取的起止位置，从而正确地截取变长的字符串。如下代码对字符型 NAME 变量明确指示从数据行前 15 个字节中读取值，而对字符型 SEX 变量则没有指定列指针，默认读取前一个变量之后到下一个变量之前之间的字符。这种方式一般用于表单化的数据读取（见程序 2-20），生成的结果如图 2-10 所示。

```
程序2-20　变长数据—按位置读取
data myclass;
  input Name $1-15Sex $ Age Height Weight;
  datalines;
Alfred Liu     M 14 112.5 69
Alice Wang     F 13 84 56.5
Barbara Deng   F 13 98 65.3
;
```

Obs	Name	Sex	Age	Height	Weight
1	Alfred Liu	M	14	112 5	69 0
2	Alice Wang	F	13	84.0	56.5
3	Barbara Deng	F	13	98.0	65.3

图 2-10　读取变长数据

（5）如果在一个数据行包括多个观测，数据呈锯齿状。我们该如何读取呢？SAS

在 INPUT 语句上设计了一个特殊选项 @@，用于告诉 SAS 从数据行完整读取一个观测后不要马上读取下一个数据行，而是继续从当前的数据缓冲区中继续读取数据填充观测。这种灵活设计为读取各种复杂格式的数据，节省 SAS 代码内嵌数据行数非常有用。比如下面的代码依然可以读取数据行中的锯齿状数据（见程序 2-21）。

程序2-21　忽略原始数据格式从缓冲区连续读取

```
data myclass;
  input Name $ Sex $ Age Height Weight @@;
  datalines;
Alfred M 14 112.5 69
Alice F 13 84 56.5    Barbara F 13 98 65.3
;
```

（6）如果数据集的一个观测来自多行文本，此时需要联合游标控制符 # 和 @ 来读取数据，它们分别表示对应数据行和列的偏移位置。此时，input 语句有若干参数可指定用来接收执行过程中所读取的文件指针信息。下面的代码中数据行格式比较混乱，我们需要一次读取 3 行文本才能获得一个观测的完整数据（见程序 2-22）。

程序2-22　利用游标控制符读取跨行数据

```
data mydata;
  infile datalines line=LN col=COL;
  input name $ 1-10
         #2 @3  Sex $ /*从第2行第三列读取一个字符串变量 Sex */
         #3 Age Weight Height; /*从第3行读取三个数值变量 */
  datalines;
Alfred
  M
14 112.5 69
Alice
  F
13 84 56.5
Barbara
  F
13 98 65.3
;
```

2.基于外部数据文件创建SAS数据集

大部分情况下，数据来自磁盘上的某个外部文件，而且通常是一系列的文件。比如在 C:\TEMP 目录中有如下文本文件 MYCLASS.TXT（见图 2-11），我们怎么用 DATA 步来读取呢？



图 2-11　待读取的文本文件

此时我们不再需要数据行语句 DATALINES，而是在 DATA 步中利用 INFILE 语句

指定该外部文件，相当于我们将 DATALINES 语句后的数据行移到了外部文件，然后再用 INPUT 语句读取。假如 MYCLASS.TXT 是包含数据的文本文件，可用如下 4 行语句进行读取数据生成了数据集 C:\temp\myclass.sas7bdat（见程序 2-23），结果如图 2-12 所示。

程序2-23 读取外部文件中的数据

```
libname mylib "C:\temp";
data mylib.myclass;
  infile 'C:\temp\myclass.txt';
  input name $ sex $ age height weight;
run;
```

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	112.5	69.0
2	Alice	F	13	84.0	56.5
3	Barbara	F	13	98.0	65.3

图 2-12 从外部文件读取的数据集

还有一种更加标准的做法是先用 filename 语句定义一个“文件引用”myfile，然后再在 infile 语句中使用该文件引用（见程序 2-24）。

程序2-24 使用filename 定义文件引用

```
filename myfile 'C:\temp\myclass.txt';
data myclass;
  infile myfile;
  input name $ sex $ age height weight;
run;
```

很多时候外部数据文件中可能包含说明文字和注释，以及数据的表头信息。这种情况下我们真正读取数据时需要忽略掉这些说明性的内容，此时可在 infile 语句上可指定读取观测的起始行 firstobs=，同时也可以用 obs= 指定需要读取的观测行数或者限定读取的观测数，此时结果数据集的总行数为 obs-firstobs+1 行。比如下面的代码读取第 2 行开始的 10 行数据。

```
infile myfile  delimiter=',' firstobs=2 obs=11;
```

如前面章节指出的，如果外部数据文件的编码与当前 SAS 会话的编码不同，就可能导致读取的数据出现乱码。此时需要我们在 SAS 代码中明确指定数据源文件的编码格式来正确读取数据，SAS 会根据指定的输入编码自动进行数据转码。比如我们要导入的文本文件为 UTF-8 编码格式的数据，则需要指定如下选项。

```
infile myfile encoding="utf-8";
```

3. 验证生成的SAS数据集

数据导入结束后一定要仔细验证结果数据集，除了前面提到的 PROC CONTENTS 和 PROC PRINT 外，SAS 也提供专门为了比较数据集的过程步。比如为了验证我们自

已创建的数据集 MYLIB.MYCLASS 和系统 SASHELP.CLASS 数据集的差异，可以调用 PROC COMPARE 来比较两个数据集的异同（见程序 2-25），其中 base= 选项用来指定基准数据集，compare= 选项指定需要比较的数据集。

程序2-25 比较两个数据集异同

```
proc compare base=sashelp.class compare=mylib.myclass;
run;
```

系统显示两个数据集基本相同，除了 SASHELP.CLASS 有数据集 LABEL 信息，SEX 列宽度为 8 字节外，两个数据集完全一样（见图 2-13）。

```
                     The COMPARE Procedure
          Comparison of SASHELP.CLASS with MYLIB.MYCLASS
                        (Method=EXACT)

                       Data Set Summary

Dataset                 Created          Modified  NVar    NObs  Label

SASHELP.CLASS  24MAY11:14:40:19  24MAY11:14:40:19     5      19  Student Data
MYLIB.MYCLASS  17JAN17:11:51:01  17JAN17:11:51:01     5      19


                       Variables Summary

              Number of Variables in Common: 5.
              Number of Variables with Differing Attributes: 1.


          Listing of Common Variables with Differing Attributes

                Variable  Dataset        Type  Length

                Sex       SASHELP.CLASS  Char       1
                          MYLIB.MYCLASS  Char       8
```

图 2-13 SAS 数据集比较

2.3.2 通过已有 SAS 数据集生成

很多时候我们都是操作已有的SAS数据集，此时可通过特定变换来生成目标数据集。比如对别人已经提供的 SAS 数据集进行增删改查以及排序、合并、分离、转置等操作来生成新的数据集。这种非分析的数据处理在数据工作中也是重要的组成部分。

（1）追加数据行：比如在 SASHELP.CLASS 数据集尾部追加一行数据生成 CLASS2 数据集，可使用 SET 语句进行（见程序 2-26），输出结果如图 2-14 所示。

程序2-26 追加数据行

```
data onerow;
  name="leon"; sex="m"; age=30; weight=83.5; height=175;
run;
data class2;
  set sashelp.class onerow;
run;
```

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
2	Alice	F	13	56.5	84.0
19	William	M	15	66.5	112.0
20	Leon	M	30	175.0	83.5

图 2-14 尾部追加数据行

也可在数据头部增加数据行，只需改变SET语句中的数据集顺序即可（见程序2-27），结果如图2-15所示。

程序2-27 头部插入数据行

```
data myclass;
  set OneRow sashelp.class;
run;
```

Obs	Name	Sex	Age	Weight	Height
1	Leon	M	30	83.5	175.0
2	Alfr	M	14	112.5	69.0
3	Alic	F	13	84.0	56.5
4	Barb	F	13	98.0	65.3

图 2-15 头部插入数据行

细心的读者可能会发现，输出的数据集中NAME有截断错误，如ALFRED变成了Alfr，原因是SET语句执行时所用的PDV初始结构来自我们创建的临时数据集ONEROW，而该数据集中变量NAME的默认长度定义来自初始值LEON, 其长度为4，不足以存储大于4字节的值，我们可以通过显示指定onerow数据集的宽度定义来修正这个问题（见程序2-28）。

程序2-28 修正默认宽度问题

```
data onerow;
  length name $8;
  name="leon"; sex="m"; age=30; weight=83.5; height=175;
run;
```

如果需要在特定行处插入数据，也可以使用内部行计数器_N_作控制实现。下面的代码在第2行插入一个观测（见程序2-29），结果如图2-16所示。

程序2-29 指定位置处插入数据行

```
data myclass;
  set sashelp.class;
  if _n_ = 1 then do;     /*在第1行后面插入数据*/
    output;
    name "leon"; sex "m"; age 30; weight 83.5; height 175;
  end;
  output;
run;
```

```
options obs 3; /*列出前3行*/
proc print data=myclass;
run;
```

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
2	Leon	M	30	175.0	83.5
3	Alice	F	13	56.5	84.0

图 2-16　指定位置插入数据行

（2）删除数据行：可以将原始数据集中的某些行数据进行剔除，形成新的数据集。下面的代码剔除了 SASHELP.CLASS 数据集 19 行数据中的第 3 行数据，生成了 18 行的数据集 myclass（见程序 2-30）。

```
程序2-30　删除第3行数据
data myclass;
  set sashelp.class;
  if _N_ = 3 then return;
  else output;
run;
proc summary data=myclass print; run;
```

当然你也可以删除满足指定条件的数据行，实际用户可以构造出任何复杂的删除逻辑（见程序 2-31），结果如图 2-17 所示。

```
程序2-31　删除性别为M的数据
data myclass;
  set sashelp.class;
  if Sex = 'M' then return;
  else output;
run;
proc print data=myclass;
run;
```

Obs	Name	Sex	Age	Height	Weight
1	Alice	F	13	56.5	84.0
2	Barbara	F	13	65.3	98.0
8	Louise	F	12	56.3	77.0
9	Mary	F	15	66.5	112.0

图 2-17　剔除 Sex=M 数据行

（3）修改数据行：满足特定条件时修改变量的值，如把第 3 行的 Name 改为“Baby”（见程序 2-32），结果如图 2-18 所示。

程序2-32　修改特定数据行

```
Data myclass;
  set sashelp.class;
  if _N_ = 3 then name="Baby";
run;
options obs=3;
proc print ; run;
```

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
2	Alice	F	13	56.5	84.0
3	Baby	F	13	65.3	98.0

图 2-18　修改第 3 行数据

（4）查询特定数据行：程序 2-33 仅输出满足特定条件的 1 行数据。

程序2-33　查询特定数据行

```
data myclass;
  set sashelp.class;
  where name="Alfred";
run;

proc print data=myclass;run;
```

2.3.3　通过 PROC IMPORT 或 PROC SQL 生成

SAS 提供 PROC IMPORT 和 PROC EXPORT 来将数据导入和导出 SAS 运行环境，如最常用的数据文件格式为逗号分隔的 CSV 和微软的电子表格 EXCEL 文件，我们可以使用如下代码完成数据的导入和导出（见程序 2-34）。

程序2-34　PROC IMPORT 导入逗号分隔文本文件

```
proc import datafile='C:\temp\class.csv' dbms=csv out=class replace;
  getnames=YES;
run;
proc print data=class; run;
```

导入 EXCEL 格式的文件，需要指定 dbms 为 xlsx 或 xls，分别对应不同版本的 Excel 电子表格数据（见程序 2-35）。

程序2-35　PROC IMPORT 导入Excel 电子表格文件

```
proc import datafile='C:\temp\class.xlsx' dbms=xlsx out=class replace;
  getnames=YES;
run;
```

上面例子中需要的数据文件 C:\TEMP\CLASS.CSV 和 CLASS.XLSX 可通过 PROC EXPORT 从 SAS 内置的数据集 SASHELP.CLASS 生成，代码如下（见程序 2-36）。

程序2-36 PROC EXPORT 导出SAS 数据集为外部文件

```
proc export data=sashelp.class outfile='C:\temp\class.csv' dbms=csv replace;
run;

proc export data=sashelp.class outfile='C:\temp\class.xlsx' dbms=xlsx replace;
run;
```

需要注意的一点是，很多人根据 SAS 帮助文档导入 EXCEL 文件时往往使用 DBMS=EXCEL 进行导入，结果发现数据导入失败并且系统会报告如下错误：

```
ERROR:Connect: 没有注册类
ERROR:Error in the LIBNAME statement.
```

根本原因是 DBMS=XLSX 和 DBMS=EXCEL 在 SAS 里访问机制是不同的，后者需要单独安装微软的 ACE 引擎才能工作，而默认情况下并没有安装它，这是一个令很多程序员感到困惑以为默认情况下 SAS 连 EXCEL 文件都不能导入的技术陷阱，其实是参数错误。

PROC SQL 是 SAS 语言中非常强大的过程步，它让用户可以在 SAS 语言中使用 SQL 语言对数据进行增删改查操作，广泛用于关系数据库管理系统的数据表和视图的增删改查。其主要功能包括：创建数据表和数据视图，对数据列作索引；查询存储在数据表和数据视图中的数据；增删改数据行或列本身，甚至将某关系型数据库特有的 SQL 语句发送到数据库管理系统中进行数据查询。另外，SAS 也支持将 SQL 查询结果置于 SAS 宏变量中，从而实现数据在数据库空间和 SAS 程序空间的传递功能，这一点非常有用。下面若干的例子来说明 PROC SQL 的用途，如创建 / 查询数据表。

（1）基于已有的数据集创建新的数据集，原来的数据集既可以是 SAS 数据集，也可以是数据库里面的表（见程序 2-37）。SAS 逻辑库对 SAS 程序员隐藏了数据库访问的细节，因此非常方便快捷。

程序2-37 基于现有数据集创建新的数据集

```
proc sql;
  create table myclass as
    select Name, Sex, Age, Height, Weight format=best.
      from sashelp.class;
proc print data=mylib.myclass; run;
```

（2）使用标准的 SQL 语言创建数据集（见程序 2-38），如果目标逻辑库 mylib 指向的是某个外部数据库管理系统，则 SAS 会自动在该数据库中创建对应的数据表。SAS 这一强大的数据库隔离功能使数据分析人员不用关心后台数据库到底是怎么存储的，不管是 Oracle、DB2 还是 SQLServer，在 SAS 看来都是一样的。

程序2-38 用 SQL语言创建数据集

```
libname mylib 'C:\temp';

proc sql;
  create table mylib.myclass( Name char(8), Sex char(1), Age num,
    Height num,Weight num informat=best. format=best.);
```

```
  insert into mylib.myclass
  values('Leon','M',31,175,80)
  values('Jim', 'M',30,173,75);

  title 'Table mylib.myclass';
  select * from mylib.myclass;
quit;
proc printto; run;
```

（3）PROC SQL语言配合宏变量在SAS中可以实现数据在不同过程步之间的传递。比如下面的代码（见程序2-39）筛选sashelp.class中体重大于平均值的那些人，首先要找到平均体重。

程序2-39　利用宏变量在过程步之间传递数据

```
proc sql; /*寻找平均体重*/
  select mean(weight) into: MEAN_WEIGHT from sashelp.class;
run;quit;

proc print data=sashelp.class( where=(weight>&mean_weight) );
/*只打印大于平均体重的学生*/
run;
```

如果要将大于平均体重的学生输出到另一个数据集，我们只需要用多行SQL语句或配合data步生成即可，代码如下（见程序2-40）。

程序2-40　在PROC SQL 内部使用宏变量传递数值

```
proc sql; /*寻找平均体重*/
  select mean(weight) into :MEAN_WEIGHT from sashelp.class;

  create table myclass as/*生成体重大于平均值的子集*/
    select * from sashelp.class where weight>&MEAN_WEIGHT;
run;quit;
```

或者如程序2-41：

程序2-41　利用宏变量在PROC SQL 和DATA 步之间传递数据

```
proc sql; /*寻找平均体重*/
  select mean(weight) into :MEAN_WEIGHT from sashelp.class;
run;quit;

data myclass;
  set sashelp.class;
  if weight>&MEAN_WEIGHT then output; /*生成体重大于平均值的子集*/
run;
```

实际上，除了前面的几种数据生成方式以外，SAS还可以根据数学函数凭空生成分析数据。SAS作为“诗一般的计算机语言”提供了极其灵活的机制，如下面的代码（见程序2-42）可以生成sin(x)和cos(x)函数的坐标数据，再调用SAS的图形过程步绘制函数曲线（见图2-19）。

程序2-42　使用数学函数生成分析数据

```
data mydata;
  do x=0 to 2 * constant("PI") by 0.1;
    sinx=sin(x);
    cosx=cos(x);
```

```
    output;
  end;
run;

title "Sin(x) and Cos(x)";
proc sgplot data=mydata;
  series x=x y=sinx;
  series x=x y=cosx;
run;
```

Obs	x	sinx	cosx
1	0.0	0.00000	1.00000
2	0.1	0.09983	0.99500
61	6.0	-0.27942	0.96017
62	6.1	-0.18216	0.98327
63	6.2	-0.08309	0.99654

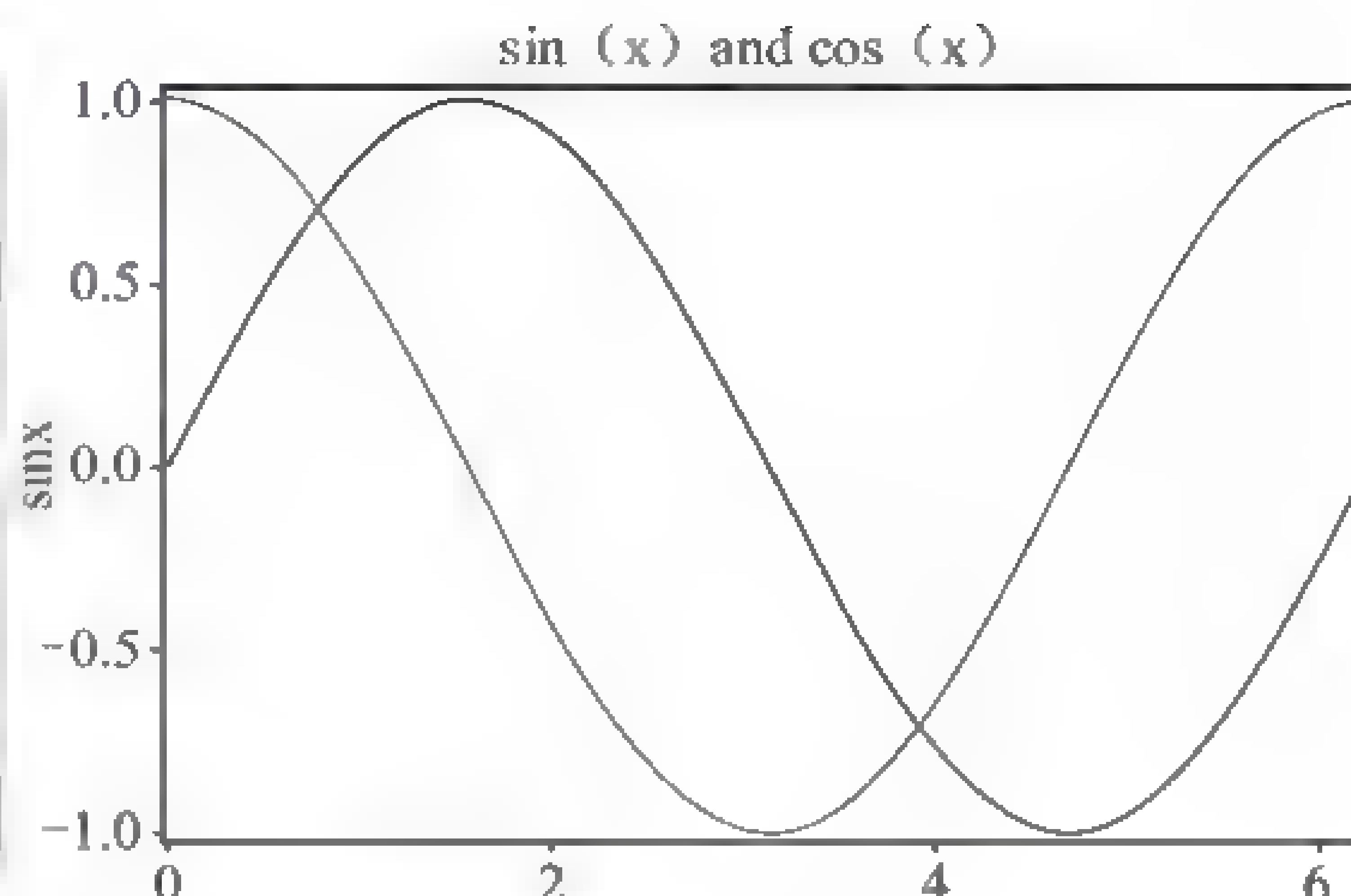


图 2-19　由函数生成数据绘图

2.4　DATA 步的运行机制

前一节的例子让我们看到 SAS 在处理数据时极其方便快捷，但这依然不够，我们还需要深入探索 SAS 的 DATA 步的工作原理，也就是需要深刻理解 SAS DATA 步的运行机制，这是 SAS 数据步编程的核心内容之一，也是精通 SAS 编程的分析人员和一般水平分析人员的重要差别；理解 DATA 步的编译运行机制与下文 SAS 特有的一个核心概念 PDV 有关，可以说“平生不识 PDV，十年 SAS 亦枉然”！

首先需要指出的是，SAS 语言是按步进行编译运行的，SAS 程序与大多数编译型计算机语言程序一样，总体上要经过编译和运行两个阶段。简要流程如图 2-20 所示。

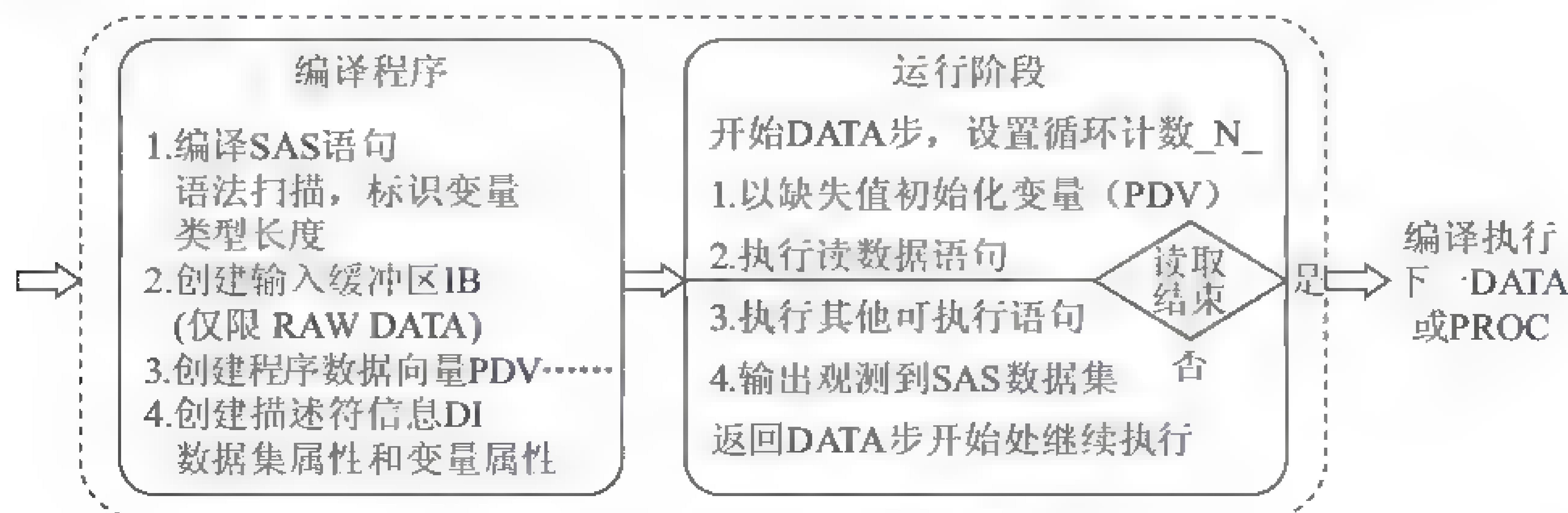


图 2-20　数据步的编译运行

2.4.1　编译阶段

编译阶段 SAS 主要做两件事。

（1）扫描 DATA 步内的每一行语句，执行语法检查和变量标识工作。编译器扫描代码片段，检查是否存在语法错误。常见的语法错误包括关键字缺失或拼写错误、无效变量名称、缺失或无效的标点符号、无效的参数或选项等。编译器也会标识每一个变量的名称、类型和长度等信息，并判断是否需要为后续变量引用作必要的类型转换。

（2）为程序执行创建必要的内部数据结构，包括输入缓冲区（Input Buffer，IB）、程序数据向量（Program Data Vector，PDV）和输出数据集描述信息（Descriptor Information，DI），其中输入缓冲区 IB 只在从外部原始数据文件中读取数据时才会创建，从 SAS 数据集中读取数据时并不需要建立输入缓冲区。

①输入缓冲区：当 DATA 步内执行 INPUT 语句是从原始数据文件（如外部文件）中读取数据记录时，SAS 会在内存中分配一块逻辑区域作为缓冲区，作为将数据放入 PDV 之前的临时缓冲区存在。如果是使用 SET 语句来读取 SAS 数据集时，SAS 则将数据直接复制到 PDV 中，而不需要所谓的输入缓冲区 IB。

②程序数据向量 PDV：当 DATA 步每读入一行数据时，都需要在内存中分配一个逻辑区域，用于存放数据集的变量和计算列信息。其数据来自于输入缓冲区 IB 或 SAS 执行语句；PDV 中还包含若干用于处理阶段的临时系统变量，它们不会被写入目标数据集。

- 行计数器_N_：用来对DATA 步的每次执行进行循环计数，从1开始；
- 错误标志_ERROR_：用来标记执行过程是否发生错误，默认值为0表示没有错误，否则为1，表示遇到一个或者多个错误。

③输出数据集描述符信息：SAS 为每一个输出数据集创建和维护的元数据信息，包括数据集属性和变量属性。比如数据集名字、成员类型、创建日期、创建时间、观测数、变量名称、类型（字符型 / 数值型）等。

下面我们考察一段 SAS 代码的编译过程来了解细节，首先需要注意 DATA 步内的 SAS 语句分成两种：一种是声明性（DECLARATIVE）语句，用于为 SAS 提供信息并在编译阶段起作用；另一种是可运行（EXECUTABLE）语句，在 DATA 步的每一次隐性循环时执行某个动作。

对于下面的例子（见程序 2-43），注释是声明性的语句在 DATA 步内的顺序不那么重要，由于它们在编译时被 SAS 解析使用，因此并不涉及执行顺序问题。然而，由于某些声明性语句引用的参数可能需要依赖于前面某条语句的编译结果，因此声明性语句依然有潜在的编译顺序问题需要考虑。比如下面的例子中，声明性语句 WHERE 就由于 AGE, SEX 变量依赖于 SASHELP.CLASS, 因此需要把 WHERE 语句放到可执行语句 SET 之后，只有编译了 SET 语句才能在 WHERE 语句中使用 AGE, SEX 变量。

程序2-43 DATA步编译运行机制

```
data myclass;
  set sashelp.class;
  where age>12 and sex='男'; /*声明性语句*/
  BMI=(weight * 0.4535924)   / ((height*2.54/100) **2);
  format BMI 4.1;            /*声明性语句*/
  label BMI="体质指数";       /*声明性语句*/
  drop weight height;        /*声明性语句*/
run;
```

上面的代码进行编译，第1行告诉SAS编译器开始一个数据步，输出数据集的名字是myclass；第2行告诉编译器从sashelp.class中读取数据；第3行则告诉SAS编译器只需要读取年龄大于12且为男性的数据；第4行告诉SAS编译器新建一个数值型计算变量BMI，即体质指数Body Mass Index，它和身高体重符合指定的数学计算关系；第5行则告诉编译器设置新建变量的输出格式是长度为4位但保留1位小数；第6行则告诉编译器设置新建变量的文本标签为“体质指数”；第7行则告诉编译器在输出数据集时剔除变量height和weight。一旦数据步编译遇到RUN语句时，SAS步宣告编译结束并自动开始执行代码。

2.4.2 运行阶段

一旦编译成功，执行阶段开始。SAS的执行主要包含如下步骤。

（1）SAS首先会从DATA语句处开始执行，如果是第一次执行，SAS会设置内部变量 _N_=1 和 _ERROR_=0，否则会对内部计数器 _N_ 自动加1，用于执行计数。

（2）默认情况下，SAS从外部原始数据文件中读取一条数据记录到输入缓冲区IB中，然后创建对应的PDV，或者直接从SAS数据集中读取一个观测直接复制到程序数据向量PDV中。SAS会以缺失值对程序数据向量（PDV）中那些由INPUT语句和赋值语句创建的变量进行初始化；DATA步的子语句INPUT和SET、MERGE、MODIFY、UPDATE都可以用来读取一条记录，但那些以SET、MERGE、MODIFY或UPDATE语句读取的变量并不会被重置为缺失值。

（3）对当前记录，执行后续的SAS可执行语句，包括赋值、计算和更新等。

（4）当执行到DATA步的最后时，隐含的SAS语句OUTPUT、RETURN和RESET被自动触发。如果DATA语句包含输出数据集名称，SAS会将当前PDV中的变量作为一个观测写入输出数据集。程序执行自动返回到DATA步开始处进行下一次隐性循环。如果SAS读取外部文件或SAS数据集结束，则整个DATA步执行终止，进入下一个DATA步或PROC步的编译执行过程。

考察前面样例代码的运行过程，执行时最重要的内存结构就是程序数据向量PDV，SAS会为需要输出到目标数据集的那些变量，创建必要的描述符信息——包括数据集中各列的名称、类型、长度、输出格式、标签等。上面例子中PDV在运行时变化简单如图2-21所示。感兴趣的读者也可以在每行语句后面插入语句PUT _ALL_；来查看PDV中各变量的变化过程。



图 2-21　DATA 步语句与 PDV 运行时变化

其中 DATA 步中的新建变量（如体质指数 BMI）首先也会以缺失值“.”进行初始化，随后在执行赋值语句时对表达式重新求值，赋给新变量然后执行下一行语句。在 DATA 步的最后，SAS 会将 PDV 中除了 HEIGHT, WEIGHT 外没有删除标志的非临时变量及其值写入目标数据集，然后控制流程再返回 DATA 步的开始处进入下一次循环。

当 SAS 进入下一次循环时，PDV 内部临时变量 _N_ 计数器会自动加 1，对于非 INPUT 语句读取的数据，SAS 会保留上次读取 PDV 的变量值直到被新读取的观测所覆盖。对于 DATA 步中的新建变量（如体质指数 BMI），SAS 会重新使用缺失值“.”进行初始化。当运行到读取数据的 SAS 语句 SET 时，SAS 会自动将输入数据集 SASHELP.CLASS 中的第二个观测读取 PDV 并重新计算变量的 BMI 值；然后在 DATA 步的最后将 PDV 中的值作为第二个观测（记录）输出到结果数据集 MYCLASS 中。最后控制流程再次返回 DATA 步的开始处，重复执行上面的处理逻辑直到源数据集或外部文件所有数据被处理完毕。

从上面的执行逻辑可以看出，DATA 步此时是自带“隐性循环”，原因是代码中用了 SET 语句。SAS 数据步在读取数据集或外部文件时的这种独特隐性循环设计，可为用户处理数据时提供了非常简洁自然的处理逻辑。比如下面几行简洁的代码就可以读取当前计算机的 ODBC 配置信息，并将每一行显示在 SAS 日志窗口。代码中看不到循环语句却自带循环处理机制（见程序 2-44）。

程序2-44　基于读取数据的隐性循环机制

```
data _null_;
  length line $ 255;
  infile "C:\windows\odbcinst.ini" delimiter='0D0A'x ;
  input line;
  put _N_ line;
run;
```

2.5　DATA 步语句快速索引

表 2-1 列出了 SAS DATA 步支持的全部子语句和功能描述，第 1 列类别为 D 表示该语句为声明性语句，否则为可执行语句。SAS 语句在功能上可分为六大类：信息语句、控制语句、动作语句、文件处理语句、窗口语句和其他语句。后面的章节中将会对部分语句进行深入展开。

表 2-1 SAS 数据步语句和功能描述

类别	语　句	描　述
信息语句：为 SAS 编译器提供关于 PDV 或输出数据集额外的信息		
D	ARRAY	定义数组元素
	数组引用	描述需要被处理的数组元素，如 a[1]=1;
D	ATTRIB	关联一个或若干个变量的属性：长度，输入 / 输出格式，标签
D	LENGTH	设置变量的存储长度属性，以字节为单位
D	FORMAT/INFORMAT	设置变量的输出 / 输入格式属性
D	LABEL	设置变量的标签属性
D	RETAIN	设置 INPUT 语句或赋值语句创建变量的值在后续隐性循环中保留不变
D	DROP/KEEP	是否将变量输出到结果数据集中？DROP 排除，KEEP 保留
D	RENAME	将变量输出到结果数据集中是将变量重命名（改变名称）
控制语句：控制 DATA 步程序的执行流程		
D	语句标签	标记程序语句的执行入口，供其他语句跳入执行时使用
	DO	与 END 语句配合标记一个语句块，相当于 C 语言的 { 符号
D	END	与 DO 语句配合标记一个语句块，相当于 C 语言的 } 符号
	IF-THEN/ELSE	条件分支语句，与传统编程语言中的条件分支类似
	SELECT	多重分支语句，执行一个或多个语句块，类似于 C 语言的多分支 switch 语句
	DO 循环（DO-TO-BY）	基于某变量执行 DO ... END 之间的语句块，相当于 FOR 循环
	DO WHILE	当条件为真时执行循环体，即 DO ... WHILE 循环
	DO UNTIL	执行循环体直到条件为真，即 DO ... UNTIL 循环
	CONTINUE	中止当前循环的执行，进入下一次循环判断
	LEAVE	跳出当前循环，相当于 C 语言的 BREAK
	GO TO	立即将程序执行跳转到指定语句标签处，也可写作 GOTO
	LINK	重定向程序执行到指定语句标签处，遇到 RETURN 返回到本 LINK 语句的下一条语句继续执行
	RETURN	将程序执行返回到上一执行点：如果是 LINK 语句跳入的则返回该 LINK 语句的下一条语句处继续执行，否则返回 DATA 步开始处进入下一次隐性循环
动作语句：执行某个动作，如赋值或调用函数，或者调用 SAS 例程		
D	WHERE	从输入数据集中选择满足特定条件的观测，否则返回 DATA 的开始处
	赋值语句 =	执行表达式并将结果赋给变量，如 foo=10;
	累加语句 +	对加号右侧的表达式求值，并将结果与左侧的变量相加后赋给左侧的变量，实现累加功能；加号左侧变量默认初值为 0，比如 sum+i;
	CALL	调用 CALL 例程，相当于调用子过程，如 call execute（…）
	OUTPUT	将当前 PDV 中没有删除标志的变量集输出到结果数据集
	PUTLOG	写 SAS 日志
	IF 取子集	只有特定表达式条件满足时才往后继续执行
	REMOVE	从数据集中删除一个观测
	REPLACE	在同样位置替换一个观测
	DELETE	停止处理当前观测

（续表）

类别	语　　句	描　　述
	STOP	停止执行当前数据步，不报告错误。多用于随机数据访问
	ABORT	停止执行当前数据步，SAS 作业或者 SAS 会话，会报告错误
	ERROR	设置 _ERROR_ 为 1，也可设置写入 SAS LOG 的错误消息
	DESCRIBE	从编译的数据步程序或数据步视图中解析源代码
	EXECUTE	执行一个编译好的数据步程序
	REDIRECT	执行存储程序时指向不同的输入 / 输出数据集
	LIST	将当前正在处理观测的输入数据写入 SAS 日志，常用于调试
	LOSTCARD	当一个观测有多个记录，SAS 处理遇到缺失或无效记录时，用于重新同步输入数据
文件处理语句：处理数据集的输入文件或者 DATA 步的输出文件		
D	DATA	开始一个数据步，并提供输出数据集，视图或程序的名称
D	CARDS/DATALINES	标明后续为数据行
D	CARDS4/DATALINES4	标明后续为包括分号的数据行，数据块结束于 4 个连续分号处
D	BY	建立特殊分组变量，控制 SET/MERGE/UPDATE/MODIFY 的运行
	INFILE	为 INPUT 语句指明输入文件
	INPUT	描述输入数据记录中的数据布局并赋给相应 SAS 变量
	INPUT （列 / 格式化 / 列表 / 命名）	从指定列读取值并赋给相应的 SAS 变量 以特定输入格式读取值并赋给相应的 SAS 变量 扫描输入数据记录，读取值并赋给相应的 SAS 变量 按名值对方式读取变量并赋给相应的 SAS 变量。
	FILE	为 PUT 语句指定输出文件
	PUT	输出文本行到 SAS 日志，SAS 输出窗口，或最近指定的外部文件的特定位置
	PUT （列 / 格式化 / 列表 / 命名）	将变量值写到外部文件特定列 将变量值以特定格式写到外部文件 将变量值和特定字符串写到特定输出行 将变量名称，等号和变量值输出
	SET	从一个或多个数据集中读取一个观测
	MERGE	将来自两个或多个数据集的观测合并为一个观测
	UPDATE	通过应用事务处理更新一个主文件
	MODIFY	替换，删除或追加记录到一个已经存在的数据集，而不创建额外复制
窗口语句：定义或显示自定义窗口，用于 SAS 代码运行的用户交互		
	WINDOW	为应用创建定制窗口用于交互
	DISPLAY	显示用 WINDOW 语句创建的定制窗口
其他语句		
	空语句	空语句，不执行任何操作
	RESETLINE	重置输出到 SAS 日志中的程序行号为 1
	FILE, ODS	用 ODS 打开特定文件
	PUT, ODS	用 ODS 向特定文件进行操作
	DECLARE	声明对象（Hash. HIter. JavaObj. ODSOUT） 以及 DS2 变量或临时数组

本章我们先介绍了利用 SAS 创建数据集的几种灵活方法，随后介绍了 DATA 步运行机制。深刻理解 DATA 步的编译运行机制对掌握 SAS 编程至关重要，灵活使用 DATA 步可为数据分析之前的数据处理提供各种变换和处理的功能。

下面以一个简单的 SAS 程序（见程序 2-45）结束本章学习，该程序用于生成黄金分割数列的前 10 项。黄金分割数列又称斐波那契数列，第 1 项和第 2 项为 1，从第 3 项开始任一项为前面两项之和；随着项数的增加，后一数与前一数的比例越来越接近于黄金比例 ϕ $(1+\sqrt{5})/2 \approx 1.6180339887$。黄金分割数列的分布在平面上呈现出极致的均衡与和谐之美，它以完整铺满整个几何平面；著名的帕斯卡三角式的浅对角线数字之和也刚好构成黄金分割数列（见图 2-22）。长宽比为 ϕ 的矩形被称为黄金矩形，据说古希腊雅典卫城的帕台农神庙的宽度和高度比接近于黄金比例 ϕ。

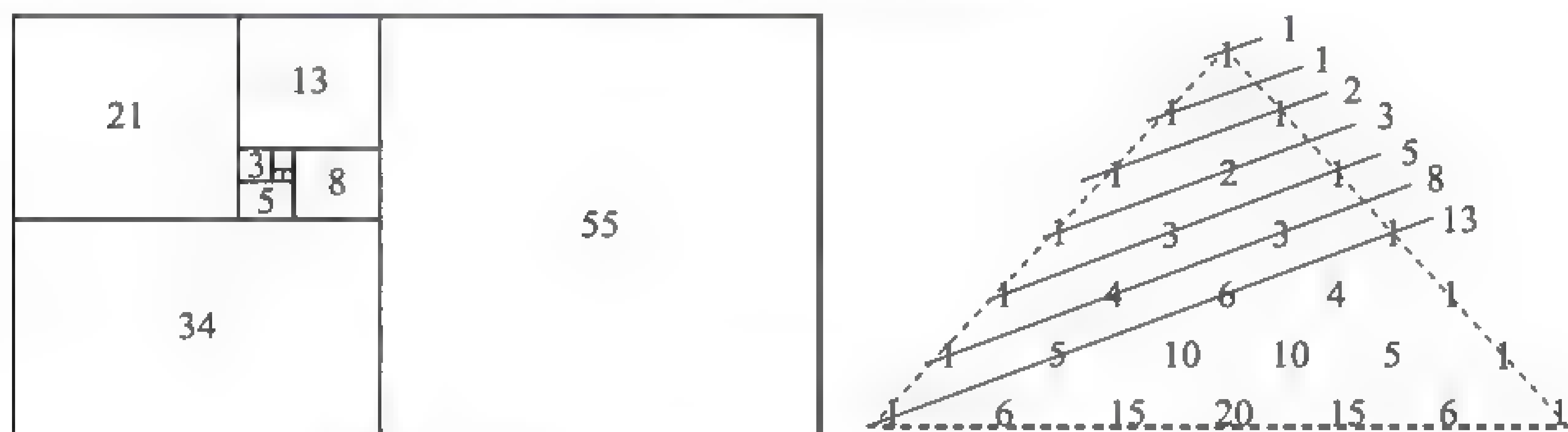


图 2-22　黄金分割数列的正方形可螺旋铺满整个平面

程序2-45　生成黄金分割数列前10 个数到 WORK.FBNC中

```
data Fbnc;
  do n=1 to 10;
    if n=1 or n=2 then x=1;
    else x=x1 + x2;
    x2=x1; x1=x;
   output;
  end;
  keep x;;
  format x best32.;
run;
proc print data=fbnc;run;
```

系统输出如下（见图 2-23）。

Obs	y
1	1
2	1
3	2
4	3
5	5
6	8
7	13
8	21
9	34
10	55

图 2-23　前 10 个黄金分割数列

通过修改参数，上面的程序最多能够计算出前 1476 个黄金分割数列，从第 1477 位开始将出现计算溢出现象。由于 SAS 能够保持 16 位有效精度，因此数列中前 78 个是毫无精度损失的计算结果。持久化输出到 SAS 数据集后从第 79 个到第 154 个数虽然系统用完整的整数表示，但其精度已经部分丢失，系统从 155 位开始使用科学计数法表示。因此，严格地说上面的程序只能处理 78 个精确表示的黄金数列。如果用 C 语言或 Java 语言实现上面的逻辑结果会更糟，要解决这种问题需要引入高精度计算机制，后面的章节中会探讨如何生成任意长度，无精度损失的黄金分割数列。

第3章

变量与表达式

3.1 常量与变量

认识世界的第一步是正确命名各种实体，在SAS程序世界中，标识符是就是用于命名编程语言实体的名称。常用的标识符分为变量名和成员名两大类，包括常量名、变量名、数组名、函数名、逻辑库和文件引用名称、成员和数据集名称等。标识符名称只能以字母或下划线开头，由字母、下划线或数字组成。大部分SAS标识符名称遵守不超过32字节的长度规则，其中：

（1）逻辑库引用 / 文件引用（Libref/Fileref）名称最长为8字节。

（2）函数 / 调用例程（Function/CALL Routines）名称最长为16字节。

（3）格式名长度限制：字符和数值型输入格式名称最长分别为30字节和31字节，字符和数值型输出格式名称最长分别为31字节和32字节。

（4）数据集的描述标签最长为32字节，但数据列的描述标签最长可为256字节。

下面的例子（见程序3-1）演示了逻辑库引用和数据集名称的命名规则，其中逻辑库引用最长只能为8个字节，而数据集名称和变量名称最长为32个字节。如果你增加一个字符则会报编译错。变量名称必须以字母或下划线开始。

程序3-1　逻辑库引用和数据集命名规则

```
libname lib45678  "C:\temp";  /*逻辑库引用名最长8字节*/

data lib45678.tab45678901234567890123456789012;/*数据集名最长 32 字节 */
  col45678901234567890123456789012=10; /*数据列(变量名)最长 32 字节 */
  a_=10;
  _a=10;
run;
```

在下面的例子中（见程序3-2），Fileref标识符最长只能为8个字符。该代码演示了如何将中文字符串以UTF-8文件编码方式写入外部文件C:\TEMP\TEST.TXT中。

程序3-2　文件引用命名规则

```
filename file5678 'C:\temp\test.txt' encoding="utf-8";

data _null_;
  file file5678;
  put "你好，SAS! ";
run;
```

SAS 中有两个系统选项用于控制变量名和成员名的命名规则：

（1）VALIDVARNAME：指定 SAS 会话中可以创建和处理的有效的 SAS 变量名，有 3 个取值可选：V7 | UPCASE | ANY，默认值为传统命名规则 V7，UPCASE 就是在 V7 命名规则之上要求变量名大写，主要用于兼容早期 SAS 版本；如果变量命名中要包含国家语言字符（如中文），则必须指定该系统选项值为 ANY。用法如下：

```
options validvarname=ANY;
```

当然，即使变量名启用了 ANY 命名规则，变量名仍然不能与系统自动变量 _N_、_ERROR_ 或者其他系统变量 _NUMERIC_、_CHARACTER_ 和 _ALL_ 命名相冲突。

（2）VALIDMEMNAME：指定有效的 SAS 成员名，所谓 SAS 成员（MEMBER）主要包括 SAS 数据集、SAS 数据视图、SAS CATALOG、SAS 索引及 SAS ITEM STORE 等数据组织单元。有 2 个可选取值：COMPATIBLE | EXTEND，默认值为 COMPATIBLE。如果成员命名中要包含国家语言字符（如中文）或除了 / \ * ? “<> | : - 10 个字符之外的特殊字符，必须指定为 EXTEND 取值。其用法如下：

```
options validmemname=EXTEND;
```

如果启用了扩展命名规则 EXTEND，我们就可以在 SAS 里使用非英文的标识符来命名数据集。当然英文句号“.”是 SAS 默认的成员分隔符，因此即使启用了 EXTEND 成员命名规则，成员名命名仍然不得以英文句号“.”开头；而对于 SPDE 引擎访问的数据，名称还要求不得包含英文句号“.”且不能以美元符 $ 开头。简单例子如下（见程序 3-3）：

程序3-3　变量名和成员名扩展命名规则演示

```
libname lib45678 "C:\temp"; /*逻辑库名最长8字节*/
data lib45678."中央人民政府数据库"n; /*需要 validmemname=extend否则编译错*/
  "中央人民政府变量名"n=10; /*需要 validvarname=ANY 否则编译错*/
run;
proc contents;run;
```

在 SAS 中可使用如下语句查找当前 SAS 会话中这两个系统选项的设置（见程序 3-4）：

程序3-4　查找当前系统选项设置

```
proc options option=(validvarname validmemname);
run;
```

3.1.1　变量长度与缺失值

对于变量，默认情况下 SAS 并不要求变量预先声明。SAS 数值型和字符型变量的默认变量长度都是 8 字节。数值型变量通常只能指定 3~8 字节的任何长度，其长度限制取决于具体的操作系统：在 Linux/Windows 平台上最小长度为 3 字节，而在 IBM 大型机 z/OS 平台上最小长度可为 2 字节。而字符型则可指定 1~32767 字节的任何长度，变量的类型和长度在 SAS 数据中可使 LENGTH、FORMAT 或者 ATTRIB 3 个语句中的任

一进行指定，其中 ATTRIB 语句可同时为多个变量指定类型和长度；FORMAT 语句则在指定类型和长度时也指定了输出格式信息，但 FORMAT 语句中指定的长度仅对字符型变量在未指定长度时有效（见程序 3-5）。

程序3-5　变量的类型和长度指定的3种方法

```
data mydata;
  /*数值型 3-8, z/OS为 2-8*/
  length n1 8;
  attrib n2 length=8;
  format n3 8.;
  /*字符型 1-32767*/
  length c1 $ 8;
  attrib c2 length=$8;
  format c3 $8.;
run;
proc contents;run;
```

SAS 作为一种专业数据处理语言，需要经常处理统计学试验或观测中没有获得的数据，称为缺失值（MISSING VALUE）。缺失值与传统编程语言中的空值 NULL 不同，它是数据分析科学中表示没有数据或观测缺失的特殊表示。数值型缺失值在 SAS 语言中用单个小数点符号“.”表示，字符型缺失值用引号括起来的单个空格字符“　”表示（见程序 3-6）。

程序3-6　SAS缺失值类型

```
data _null_;
  missingnum=.;  *指定数值型缺失值;
  missingstr=' '; *指定字符型缺失值;
  *call missing(missingnum, missingstr);/*该CALL 例程用于将变量置为缺失值*/

  nullstr=""; *对于字符型变量，传统的空字符也被当作缺失值看待;

  *检测变量是否为缺失值;
  if missingnum=. then put "missingnum is missing";
  if missingstr=' ' then put "missingstr is missing";
  if nullstr=' ' then put "nullstr is missing";

  *使用missing 函数来检查缺失值;
  if missing(missingnum) then put "missingnum is missing";
  if missing(missingstr) then put "missingstr is missing";
  if missing(nullstr) then put "nullstr is missing";

  length dotstr $ 1; *对于字符型变量，赋给 . 等价于赋给 ".";
  dotstr=.;
  if dotstr='.' then put "dotstr is equal to '.'";
  if missing(dotstr)=0 then put "dotstr is NOT missing";
run;
```

系统输出：

```
missingnum is missing
missingstr is missing
nullstr is missing
missingnum is missing
missingstr is missing
nullstr is missing
dotstr is equal to '.'
dotstr is NOT missing
```

3.1.2 数值常量

在 SAS 语言的 DATA 步中，可在表达式中直接定义一个数值型或字符型常量。SAS 变量存储本身不区分整型与浮点型，它是一个 8 字节有符号双精度浮点数。在 SAS 代码中用整型和浮点型常量表达式给变量赋值的时要特别注意精度控制范围。

● 整数和浮点数常量表达式

数值型常量可以用整数、定点实数以及科学计数法表示。当用户将一个数值常量表达式赋给某个变量时需要特别注意其精度保持范围。默认情况下，无精度损失的最小和最大整数区间为 9007199254740992 到 9007199254740992。它们对应的十六进制表示为 FFE0000000000000 – 0020000000000000。超过此范围的整数常量表达式赋值会有隐性精度损失，用户可以执行如下程序（见程序 3-7）来对比不同运行结果。

程序3-7　DATA步的数值型常量

```
data _null_;
  intc_min=-9007199254740992; intc_max= 9007199254740992;
  put "Integer Constant:  " intc_min 17. " ~ " intc_max 17.;

  *区间内的数值;
  intc_min2= intc_min + 1;      intc_max2= intc_max - 1;
  put "Internal Constant:  " intc_min2 17. " ~ " intc_max2 17.;

  *区间外的数值：精度损失...;
  intc_min2= intc_min - 1;      intc_max2= intc_max + 1;
  put "External Constant:  " intc_min2 17. " ~ " intc_max2 17.;
  put;

  *区间外的数值：精度损失;
  intc_min3=-9007199254740993; intc_max3= 9007199254740993;
  put "External Constant:  " intc_min3 17." ~ " intc_max3 17.;

  *区间外的数值重新回到区间内;
  intc_min4= intc_min3+1; intc_max4= intc_max3-1;
  put "Internal Constant:  " intc_min4 17." ~ " intc_max4 17.;
run;
```

系统输出：

```
Integer  Constant:  -9007199254740992 ~  9007199254740992
Internal Constant:  -9007199254740991 ~  9007199254740991
External Constant:  -9007199254740992 ~  9007199254740992

External Constant:  -9007199254740992 ~  9007199254740992
Internal Constant:  -9007199254740991 ~  9007199254740991
```

浮点数常量表达式包含小数点，它支持除小数点外共 11 位有效数字，包括符号位。如果用科学计数法表示，除小数点外的整个数字的有效位为 11 位，包括尾数、符号 E 以及 E 后的符号位和指数等内容。SAS 浮点数可表达的数值范围为：± 2.22507E-308 到 ± 1.797693E308，在 SAS 内部对应的十六进制存储表示为 000FFFFE2E8159D1 到 7FEFFFFFD7B9609A。最小最大的正数浮点数也可由系统函数 constant ("SMALL") 和 constant ("BIG") 返回。程序 3-8 可说明 SAS 的浮点数常量精度范围。

3.1.3　日期/时间/日期时间常量

在 SAS 中日期 / 时间 / 日期时间是 3 种特殊的数值型数据，其常量定义采用双引号或单引号后加 d、t、dt 或 D、T、DT 进行表示（即 Date、Time 和 DateTime 的缩写）。SAS 中的日期时间定义与别的厂家不同。它的 DateTime 定义是从公历 1960 年 1 月 1 日 00:00:00 开始的秒数，而微软对于 DateTime 的定义则是从公历 0001 年 1 月 1 日 00:00:00 开始的嘀嗒数（在计算机内 1s 等于 1000 万个嘀嗒数）。程序 3-10 演示了如何利用日期常量，时间常量以及日期时间常量对变量进行赋值。

程序3-10　Data步的日期/时间以及日期时间常量

```
data _null_;
  d='02JAN1960'd;
  t='00:00:01't;
  dt='01JAN1960 00:00:01'dt;

  put " d=" d date.;
  put " t=" t time.;
  put "dt=" dt datetime.;

  put "value=" d t dt;
  if d=t AND d=dt then put "SAME INTERNAL VALUE";
run;
```

上面的代码运行结果表明，d、t 和 dt 不指定格式时的输出值都等于 1，但它们的语义却完全不同：变量 d 表示的是时期 1960 年 1 月 2 日，变量 t 表示的是时间 00:00:01，而变量 dt 表示的是 1960 年 1 月 1 日凌晨 00:00:01。

3.1.4　字符常量

字符常量需要使用单引号或者双引号括起来，两者的区别是单引号字符串不进行宏展开，而双引号中的字符串则会进行宏展开。宏展开是 SAS 宏处理器在编译时对该字符串进行宏解析，生成最终的 SAS 代码供 SAS 编译器进行编译的过程。字符串常量中字符大小写是敏感的，而 SAS 代码中的关键字和标识符大小写是不敏感的。

考察如下代码（见程序 3-11）。

程序3-11　字符常量以及双引号中的宏变量

```
data _null_;
  c="&sysver";
  put "SAS Version is " c;

  c2='&sysver';
  put "SAS Version is " c2;
run;
```

输出结果如下，其中第一行正确获得了当前 SAS 的版本号，而第 2 行则返回原始字符串。

```
SAS Version is 9.3
SAS Version is &sysver
```

在 SAS 中如果变量名称要包含空格或者国家语言特定的字符（如中文），则我们可以使用 Name Literal 来命名它。其定义形式就是一个单引号或双引号括起来的字符串常量，在引号后面加上 n 或者 N 即可。SAS 支持这种特殊的变量命名是由于它需要跟外部的存储系统，如关系数据库 DBMS 的表名和列名打交道，因此需要在 SAS 代码中做特殊处理进行映射。注意 Name Literal 命名方式需要启用系统选项 VALIDVARNAME=ANY。考察如下代码（见程序 3-12）。

程序3-12 Name Literal 命名方式

```
options validvarname=ANY; /*启用变量任意命名选项*/

data _null_;
  pi=3.1415926;
  c="PI"; c2="pi"n;         /*变量 c2 从变量 pi 中取值*/
  put  c "=" c2;

  "var 1"n = 100;           /*双引号方式，小写 n*/
  put "var 1=" "var 1"n;

  '变量1'N = 100;           /*单引号方式，大写 N*/
  put "变量1=" '变量1'N;
run;
```

系统输出如下：

```
PI =3.1415926
var 1=100
变量 1=100
```

任何编程都是从常量和变量开始构建复杂的程序世界。SAS 语言与传统编程语言不同之处在于它生而为数据分析科学而设计，它从根上定义了面向分析的数据类型和表达方式。尽管 SAS 保留了对字节和比特位的操作能力，但 SAS 编程人员的视角更多地是基于一种面向数据分析和统计学语义的数据表达，而不是基于字节和位操作之上的面向机器数据结构和算法体系的视角。

3.2 表达式

在 SAS 语言中，SAS 表达式是 SAS 语句的组成部分，是构成 SAS 程序的一系列操作数和运算符的组合。其中操作数包括常量和变量，运算符则包括算术运算、关系运算、逻辑运算等类型。SAS 表达式也包括赋值运算及一些 SAS 特有的运算类型。

3.2.1 运算符

（1）算术运算符：用于数值常量或变量的算术运算（见表 3-1），包括加法 +、减法 、乘法 *、除法 和乘方 ** 等运算。算术运算符的优先顺序为“乘方优先，先乘除后加减”；在表达式中可用小括号来人为限定运算的优先顺序（见程序 3-13）。

表 3-1　算术运算符

运算符	+	-	*	/	**
释 义	加法	减法	乘法	除法	乘方

程序3-13　算术运算范例
```
data _null_;
  a= 3; b=4; c=5; d=6;
  d = a + b;
  e = a * b;
  f = a / b;
  g = a ** 2;
  h = a + b  * c  / 6;
  i =(a + b) * c  / 6;

  put "d=" d "e=" e "f=" f "g=" g "h=" h "i=" i;
run;
```

输出结果为 d=7 e=12 f=0.75 g=9 h=6.3333333333 i=5.8333333333

（2）关系运算符：用于比较常量或变量是否相等，以及它们的大小关系（见表 3-2）。结果为真返回 1，否则返回 0。

表 3-2　关系运算符

运算符	=	^=	<	<=	>	>=
释 义	等于	不等于	小于	小于等于	大于	大于等于

各种关系运算符也可以分别使用对应的别名 EQ、NE、LT、LE、GT、GE 表示，其中不等于、小于等于和大于等于还可使用早期变体 ~=、=< 和 => 表示。数值型数据的比较基于数值大小进行；而字符型数据则基于字符的码点值大小进行比较，SAS 规定缺失值和空格字符在比较运算中比任何可打印字符都小。为了跟早期 SAS 版本兼容，SAS 表达式中的不等于运算符 ^= 也可以写作 ~=，而小于等于 <= 和大于等于 >= 也可以写作 =< 和 =>，不过现在已经很少人使用这种写法。考察如下代码（见程序 3-14）。

程序3-14　关系运算范例
```
data _null_;
  a= 3; b=4; c=5; d=6;

  e=(a ^= b); f=(a ~=b );
  put "(a^=b)=" e "(a~=b)=" f;

  g=(a => b); h=(a =< b );
  put "(a=>b)=" g "(a=<b)=" h;

  if d GT c then put "Alias work";
run;
```

输出结果为

```
(a^=b)=1 (a~=b)=1
(a=>b)=0 (a=<b)=1
Alias work
```

传统的C/C++语言中的等于和不等于运算符为 == 和 !=，而SAS依然使用单字符 = 和双字符 ^= 来表示等于和不等于运算。此外，SAS语言还支持一些特殊的关系表达式写法，如SAS可以把变量写在中间，变量前后都是运算符，比如：3<=x<=5，它对应的别名写法就是3<=x AND x <=5，表示x介于3和5之间。

（3）逻辑运算符：用于表达式中的逻辑运算，包括逻辑与、逻辑或和逻辑非3种运算（见表3-3）。也可使用对应的别名AND、OR和NOT表示，其中逻辑或也可用早期变体 ¦ 或 ! 表示。

表3-3 逻辑运算符

运算符	&	|	^
释 义	逻辑与	逻辑或	逻辑非

在各种编程语言中，通常假（False）用0表示，真（True）用1表示。而在SAS程序中0或缺失值都被视为假，而任何非0值和非缺失值则被视为真。因此，SAS程序中单个数值变量默认可参与逻辑运算，只要它不是0或缺失值都会被视作真（见程序3-15）。

程序3-15 逻辑运算范例

```
data _null_;
  a= 3; b=4; c=5; d=6;

  e =(a<b)|(c>d);
  e2=(a<b)¦(c>d);  e3=(a<b)!(c>d);

  f= (a<b) & (c>d);
  g=^(a<b);

  if a then put "a is true";

  put "or=" e "and=" f "not=" g;
  put "or=" e2 "or=" e3;
run;
```

输出结果为

```
a is true
or=1 and=0 not=0
or=1 or=1
```

与C/C++的逻辑运算符不同，SAS的“逻辑与”跟“逻辑或”采用单字符表示，而且感叹号 ! 符在SAS中是逻辑或，而在C/C++中是逻辑非操作；双竖线符号 || 在SAS中是字符串拼接操作，而在C/C++中是逻辑或运算。

除上面列出的运算符外，SAS语言还有一些其他语言没有的特殊运算符（见表3-4），包括：

表3-4 特殊运算符

运算符	<>	><	IN	||
释 义	取最大值	取最小值	属于	字符串拼接

取最大值 / 最小值运算符为 <> 和 ><，相当于 MAX/MIN 函数，但在 SAS 语言中它被当作一种基本运算符存在。SAS 支持多种缺失值，如果特殊缺失值是表达式中比较运算的一部分，SAS 则会用缺失值的排序顺序进行比较，即以小数点后的字母进行比较，比如缺失值 .A <> .Z 返回 .Z（见程序 3-16）。

程序3-16　取最大/最小值运算范例

```
data _null_;
  max= 3<>5;
  min= 3><5;
  max_missing= .z<>.a;

  put "Max=" max "Min=" min "MAX_MISSING=." max_missing;
run;
```

输出：

```
Max=5 Min=3 MAX_MISSING=.Z
```

注意：由于部分计算机语言使用 <> 作为不等于运算符，因此需要警惕该运算符在 SAS 表达式中可能引起的歧义。程序 3-17 并不是输出 TRUE，而是输出 FALSE。由于在 IF 语句里 x <> y 返回的是 x，而 x=0 在 SAS 里是表示假，因此输出 FALSE，它也不表示变量 x 等于变量 y。

程序3-17　一个可能的“比较运算”陷阱

```
data _null_;
  x=0; y=-1;
  if x <> y then put "TRUE";
  else put "FALSE";
run;
```

集合运算符 IN 是 SAS 语言特有的运算符，用于检查变量的值是否在一个给定的列表中。列表内必须为常量、缺失值或遍历器。考察如下例子（见程序 3-18），变量 b、d 和 f 都会输出 1。

程序3-18　集合包含运算

```
data _null_;
  a= 3;
  b= a IN (3 , 4, 5, 6 );

  c="Jane";
  d=c IN ("Tony", "Jane", "Fox");

  array e[3] (1, 2, 3);
  f=a IN e;

  put b= d= f=;
run;
```

字符拼接运算符 || 用于连接两个字符型变量或常量，但该运算符并不删除前一个字符串尾部的空格，用户需要使用 trim 函数删除尾部空格再拼接字符串才能得到期望的结果（见程序 3-19）。由于 SAS 字符变量为定长字符型，它有长度定义且尾部会填充空格补齐，因此 trim 函数常常是必须的。

程序3-19　字符串拼接

```
data _null_;
  length a b $ 8;
  a="Hello";
  b="World";
  c=trim(a) || " " || b;
  put c;
run;
```

3.2.2　运算符优先顺序

当一个表达式包含多个运算符时，如果没有显示使用括号来分隔子表达式时，就会涉及运算符优先顺序问题。与其他计算机语言一样，表达式中的括号 ( ) 具有最高优先级，用户也可以使用括号来分隔表达式指定结合顺序，避免潜在的歧义性。

表 3-5 为 SAS 运算符优先顺序的完整列表，其中乘方、取最大 / 最小值，取正负号运算符、逻辑非 6 种运算符为第一优先级且从右到左进行求值，其他的运算符都是从左到右进行求值。编程实践中鼓励大家明确使用括号来分隔表达式，或者将一个复杂表达式分解为多个赋值语句来提高代码的可读性。

表 3-5　SAS 运算符优先级

优先级	运算符	类别	释义
0	( )		括号指定结合顺序最优先
1 此组运算符按从右到左求值	**	算术运算	乘方运算
	+ −	单目运算	对右值取正 / 负号
	^	关系运算	逻辑非
	<> ><	SAS 特定	取最大值、最小值运算
2	× /	算术运算	乘除法
3	+ −		加减法
4	||	SAS 特定	字符串拼接
5	< <= = ^= >= >	关系运算	小于、小于等于、 等于、不等于、 大于等于、大于
	IN	SAS 特定	集合包含运算
6	&	逻辑运算	逻辑与
7	|	逻辑运算	逻辑或
8	=	赋值运算	

3.2.3　WHERE 语句特定运算符

在 DATA 步的 WHERE 语句中包含一些特殊的运算符如 LIKE 和 IS 运算符（见表 3-6），但它们只能在 WHERE 语句中使用，作用范围非常有限。

表 3-6　WHERE 语句特定运算符

运 算 符	LIKE	IS MISSING	IS NULL
释 义	字符串通配符 % 表示任意字符串 _ 表示任意单个字符	判断变量 是否为缺失值	判断变量 是否为空值

字符串通配符比较 LIKE 用于表达式中模糊匹配字符串，如 WHERE 语句中的 LIKE 字符串比较，其中百分号“%”表示任意字符串（见程序 3-20），而下划线“_”表示任意单个字符（见程序 3-21）。

程序3-20　字符串模糊匹配

```
data myclass;
  set sashelp.class;
  where Name like "%丽%";
run;
proc print;
run;
```

输出如图 3-1 所示。

Obs	Name	Sex	Age	Height	Weight
1	爱丽丝	女	13	56.5	84
2	玛丽	女	15	66.5	112

图 3-1　任意字符模糊匹配查找

程序3-21　单个字符模糊匹配

```
data myclass;
  set sashelp.class;
  where Name like "_丽";
run;
proc print;
run;
```

输出如图 3-2 所示。

Obs	Name	Sex	Age	Height	Weight
1	玛丽	女	15	66.5	112

图 3-2　单个字符模糊匹配查找

对于运算符 IS MISSING 和 IS NULL，可在 IS 前面加上 NOT 运算符取反，如 *WHERE NAME NOT IS MISSING* 或 *WHERE NAME NOT IS NULL* 等。

3.2.4　赋值语句

赋值语句用于对一个表达式进行求值，并把结果赋给一个新变量或者已有的变量。

赋值语句是极少数不需要以关键字开始的 SAS 语句，其基本语法如下：

```
VAR = EXPRESSION;
```

等号前后分别为变量名和表达式。右侧表达式在求值时，需要根据 SAS 运算符优先顺序进行求值。如果右侧表达式中包含缺失值，则计算的结果为一个缺失值。当表达式右侧包含左侧变量时，变量首先会在求值过程中先被使用，最后才将结果存储在左侧的变量中。考察如下代码（见程序 3-22）。

程序3-22 赋值语句范例

```
data _null_;
  a=a+1;
  b=3;
  b=b+1;
  put a= b=;
run;
```

系统不会输出 a=1，而是输出 a=. 缺失值。因为变量 a 第一次出现在等号右侧时为缺失值，如果右侧表达式中包含缺失值，则计算结果为缺失值。对于变量 b，则会正常输出 b=4。

3.2.5 累加赋值语句

SAS 步中还有一个很特殊的变量累加赋值语句，它不需要关键字也不需要等号，它也是极少数不需要以关键字开始的 SAS 语句，其语法如下：

```
VAR+EXPRESSION;
```

由于只有数值型变量才能加减运算，因此变量 VAR 会被 SAS 默认为是一个数值型变量并将它初始化为 0。然后在每次执行该语句时都会触发表达式 *EXPRESSION* 重新求值后与 VAR 变量累加并赋给 VAR 变量。累加赋值语句左侧的变量 VAR 在 DATA 步隐性循环中不会被自动清零，即相当于该变量默认已经使用 RETAIN 语句进行声明。考察如下代码（见程序 3-23）。

程序3-23 累加赋值语句

```
data _null_;
  set sashelp.class;
  c+1; /*等价于 retain c 0; c=c+1*/
  put c=;
run;
```

系统输出 c=1 c=2 … c=19。

默认累加赋值语句初始值为 0，但用户可显式使用 RETAIN 语句指定其初始值。下面的代码会输出 C=101 C=102 … C=119（见程序 3-24）。

程序3-24　指定初始值的累加

```
data _null_;
  set sashelp.class;
  retain c 100;
  c=c+1; /*也可写成 c=sum(c,1)*/
  put c=;
run;
```

3.2.6　RETAIN 语句

RETAIN 语句可以让 INPUT 语句或赋值语句创建的变量在 DATA 步的隐性循环中得以保持，避免下次隐性循环时被重新用缺失值初始化。它是声明性语句而非执行性语句。其基本形式为：

```
RETAIN VAR1 VALUE1 VAR2 VALUE2 … VARN VALUEN;
```

该语句可将每个变量初始化为指定的值，如 retain a 1 b 2 c 100；也可以同时将多个变量初始化为某个特定值，如 retain a b c 100。

RETAIN 语句还支持对系统变量列表 _NUMERIC_ 进行赋值，它表示将当前 PDV 中的全部数值型变量初始化为特定值，如 RETAIN _NUMERIC_ 100; SAS 支持 _ALL_、_NUMERIC_ 和 _CHAR_ 3 个系统变量，分别表示 PDV 中的全部变量，全部数值型变量或者全部字符型变量。

RETAIN 语句只可用于除 _N_ 和 _ERROR_ 外的任何变量，如果一个变量仅仅出现在 RETAIN 语句中从未被赋过值，该变量默认不会被输出到目标数据集中。RETAIN 语句常用于指定在 DATA 步隐性循环中不要重新初始化为缺失值的那些变量。

RETAIN 语句与 KEEP 语句不同，KEEP 语句是用来指定需要“保留”输出到目标数据集中的那些变量，与之对应的 SAS 语句是 DROP 语句，表示不要将指定变量输出到目标数据集中。

考察如下代码（见程序 3-25），可以对 SASHELP.CLASS 的所有年龄求和，其中 TOTAL 能得到正确值，而 TOTAL2 只有第一个观测有值，其他都是缺失值。分析其原因是变量 TOTAL2 在下次隐性循环时会被系统默认设置为缺失值，从而导致等号左侧的 TOTAL2 也被设置为缺失值。

程序3-25　RETAIN语句范例

```
data _null_;
  set sashelp.class;

  retain total 0; /*初始化为 0*/
  total=total+ Age;

  if _N_=1 then total2=0;/*total2 首先会被初始化为缺失值，执行到此赋 0*/
  total2=total2+Age; /*total2 第二次执行会被初始化为缺失值，结果为缺失值*/

  put _N_= age= total= total2= ;
run;
```

系统输出：

```
_N_=1 Age=14 total=14 total2=14
_N_=2 Age=13 total=27 total2=.
...
_N_=19 Age=15 total=253 total2=.
```

要修正上面的问题，添加 RETAIN TOTAL2 语句即可。在 SAS 数据步中，某些变量系统默认具有在隐性循环之间保持数值的特性，也就是说它们并不需要用 RETAIN 语句显式指定就可避免在下次隐性循环开始时被重置为缺失值，默认具有“变量保持”特性的变量包括。

（1）累加求和变量。

（2）自动变量 _N_、_ERROR_ 以及 _I_、_CMD_ 和 _MSG_。

（3）SET、MERGE、MODIFY 和 UPDATE 语句中读取的变量以及这些语句中 END= 和 IN= 选项创建的变量。

（4）FILE、INFILE 语句中使用 BY= 选项创建的变量。

（5）临时数组中指定的数据元素。

（6）ARRAY 语句中初始化的数组元素，或这些数组元素的成员。

在对数据集中的观测进行遍历查找时，我们经常会用到 RETAIN 语句。灵活应用它配合隐性循环可完成一些复杂功能。比如下面的例子（见程序 3-26）查找 sashelp.class 中男生和女生的最小身高值。

程序3-26 查找特定分组内的最小值

```
proc sort data=sashelp.class out=class_sort;
  by sex;/*按照性别分组*/
run;

data class_maxmin;
  set class_sort;
  by sex;

  retain Shortest; /*分别在两组内寻找最小值*/
  if first.sex then Shortest=height;
  Shortest=Shortest >< height;/*取较小的一个*/
  if last.sex then output;

  keep sex Shortest;
run;

proc print data= class_maxmin;
  title "Shortest student";
run;
```

系统输出如图 3-3 所示。

Obs	Sex	Shortest
1	F	51.3
2	M	57.3

图 3-3 分组寻找极值

SAS 程序中表达式是构成 SAS 语句的基础，而 SAS 语句则是构成 SAS 程序的基础。就像人类语言中表达思维的文本分成段落、句子一样，表达式是 SAS 语句的一个片段，SAS 语句才是构成完整语义的最小单位。良好的表达式应该变量命名清晰，运算关系明确，没有歧义且易于维护。

3.3　SAS 数组

SAS 数组与传统编程语言的数组的本质不同在于 SAS 数组并非一种特殊的数据结构，而是一种变量的组织引用方式。SAS 数组本质上是对 SAS 变量的分组引用，因为 SAS 数组本身甚至不是一个变量，只是对一系列真正的 SAS 变量进行统一引用的名称而已，DATA 步中的 SAS 数组也被称为变量数组（Variable Array）。

SAS 数组使用 ARRAY 语句进行定义，默认情况下数组名称、数组长度是必须的，其余部分则是可选的（尖括号部分）。SAS 数组定义的完整语法如下所述。

```
ARRAY 数组名称 [数组长度] <$> <元素长度> 元素变量列表 <(元素初始值列表)>;
```

- 数组名称：任何有效的SAS名称，最大长度不得超过32字符。
- 数组长度：数组所能包含的元素个数。SAS数组的最大长度跟SAS数据集能支持的最大列数有关。在SAS 9.1 以前的版本中数据集的最大列数为32767，在更高版本中则取决SAS所在的运行环境和文件属性，即最大变量数取决于PDV中所有变量所占用字节长度的总和，其总和不得超过文件系统的最大页面大小。笔者在Windows平台上成功创建过400万列的SAS数组，它对应一个具有400万列的数据表。
- 元素类型：可选项，$ 表示数组元素类型是字符型，否则默认表示数组元素为数值型。
- 元素长度：可选项，数组中元素变量的长度，数组所有变量共享相同长度，以字节为单位。
- 元素变量列表：构成数组的变量列表，通常用VAR1 - VARN形式定义。
- 元素初值列表：依次指定数组元素的初始值，以逗号或空格分隔。

（1）由于 SAS 数组很灵活，其最常见的定义方式如下（见程序 3-27），数组元素不仅能够定义类型，也可指定元素的长度。数组中所有元素拥有同样的长度定义。比如：

程序3-27　定义数组大小和长度，以及数组元素的变量长度

```
data _null_;
  array array_n [3];       /*定义了一个具有 3 个元素的数组，元素类型缺省为数值型*/
  array array_c [3] $;     /*定义了一个具有 3 个元素的数组，元素类型为字符型*/
  array array_cx [3] $ 2; /*具有 3 个元素，元素类型为字符型，长度为2字节*/
run;
```

该代码分别定义了若干 3 个元素的数值型 / 字符型数组，系统内部会默认生成一

系列名称为ARRAY N1、ARRAY N2、ARRAY N3 和ARRAY C1、ARRAY C2、ARRAY C3的SAS变量，这些变量保存在PDV中，因此在SAS DATA步中也可以直接用变量名进行访问。数组的初始化示例见程序3-28所示：

程序3-28 数组的初始化

```
data _null_ ;
  array array_n [3];
  array_n[1]=1;  array_n[2]=2; array_n[3]=3;

  array array_c [3] $;
  array_c[1]='a'; array_c[2]='b'; array_c[3]='c';

  put array_n1= array_c1=; /* 默认生成一系列变量 array_n1…能正确工作*/

  array array_cx [3] $ 2;/*定义为字符型，元素字节长度为2*/
  array_cx[1]='ABC'; array_cx[2]='BC'; array_cx[3]='CD';
  put array_cx[1]=; /*由于指定了长度 2，ABC赋值时发生截断仅保留 AB*/
run;
```

（2）由于SAS数组本质上是变量的组织引用方式，因此它与传统的C/C++语言不同，用户可人为指定数组所映射的元素变量名称序列，如var1、var2、var3。

```
array myarray [3] var1-var3;/*默认为myarray[3]生成变量var1, var2, var3*/
```

指定的数组长度必须与所映射的变量列表具有的相同的元素数量，否则SAS会报与如下类似的语法错误。

```
ERROR：为数组 myarray 指定的维定义的变量过多。
ERROR：为数组 myarray 指定的维定义的变量过少。
```

如果用户已经在数组所映射的变量列表中隐性指定了元素的个数（比VAR1-VAR3，表示数组长度为3），用户可以不必在中括号[ ]中指定数组长度，而是用*号代替或者干脆忽略整个中括号[*]部分。如下所示：

```
array myarray1 [*] var1-var3;
array myarray2     var1-var3;
```

注意：上面两行代码虽然创建了2个数组MYARRAY1和MYARRAY2，但其实它们映射的SAS变量集是一样的。因此在任何一个数组中对数据进行了改动，实质上都是对变量VAR1-VAR3进行的修改。这种机制为相同变量集合建立不同数组映射成为可能。

（3）在数组元素变量列表后，用户也可以使用小括号来指定数组元素的初始值，元素之间使用逗号或者空格隔离。注意，在C/C++语言中使用大括号{}指定初值，而SAS使用小括号()指定。

```
array myarray[3] $ 4 var1-var3 ('ABCD','BCDE','CDEF');
```

为了理解SAS数组的运行机制，可运行如下代码（见程序3-29）来检查不同引用方式对变量值的修改。

程序3-29　数组为虚，PDV变量为实

```
data mydata;
  array myarray[3] $ 4 var1-var3 ('ABCD','BCDE','CDEF');
  var1="12345"; /*直接修改变量*/
  put myarray[1];

  myarray[1]="QWER"; /*修改数组引用*/
  put var1;
run;
```

输出结果为

```
1234
QWER
```

3.3.1　数组名称

SAS 数组必须先定义后使用，并且数组只能在当前 DATA 步内有效，由于它所映射的 SAS 变量属于特定 DATA 步的 PDV，因此数组不能跨 DATA 步使用。SAS 数组命名必须遵循如下规则：

- 数组名必须遵循SAS变量名命名规则，最长不得超过32个字符。
- 数组名不得与同一DATA步内的其他任何已有变量名冲突。
- 数组名应避免与SAS系统函数/用户自定义函数重名　如果数组名和函数名重名，该名称会优先被SAS当作数组名进行优先处理，因此SAS数组名可以阻塞对同名函数的调用。
- 数组定义和引用可以使用方括号[ ]，小括号( )或大括号 {}　虽然大括号 {} 在SAS语法中没有被别的语法定义，笔者建议使用C/C++普遍采用的中括号 []方式进行引用更符合通用代码书写习惯。

3.3.2　数组元素变量列表

由于数组能组织相同类型的变量，因此在定义数组元素变量列表时，用户可使用系统变量 _NUMERIC_ 或 _CHARACTER_ 将 PDV 中的所有的数值型或字符型变量，映射到某个数组中。比如下面的代码（见程序 3-30）可以将 SASHELP.CLASS 的字符型变量和数值型变量映射到两个数组中。

程序3-30　基于系统变量建立数组映射

```
data _null_;
  set sashelp.class;
  array array_c _CHARACTER_;/*NAME SEX*/
  array array_n _NUMERIC_; /*AGE WEIGHT HEIGHT*/
  put _ALL_;
run;
```

3.3.3 数组长度

数组长度是指数组中的元素个数，定义时可用整型数值进行指定；当长度不确定或未知时，也可以使用星号 * 进行指定。SAS 数组长度不支持使用整型变量进行动态指定（注：但可用宏变量指定，而宏变量可动态改变）。

如前所述 SAS 数组最大长度不是 32767，而是取决于 SAS 运行环境的内存配置。笔者曾在个人机环境上成功创建 2^22-1 个元素，即 4194303 列的数据集；在尝试 2^23-1（即 8388607 约 800 多万列）时系统仅报告内存不足而没有报告任何语法错误。也就是说 SAS 数组最大长度只取决于操作系统的在内存管理方面的限制。

SAS 系统函数 DIM 可返回数组特定维度的长度，如果返回第 2 个维度的长度可使用 DIM2（…）或 DIM（…, 2）函数，以此类推。比如：

```
do i = 1 to DIM(myarray);
  put myarray[i];
end;
```

用户可以自定义 SAS 数组的下标区间，一般情况下我们推荐使用默认下标索引，除非有时为了实现特定变量映射或算法需要而改变下标区间。SAS 数组下标默认是从 1 开始的，其内部映射的变量默认也是从 1 开始的。即使我们改变数组的索引范围，在 PDV 中的具体变量名称也是从 1 开始命名的。运行如下代码（见程序 3-31）可以看到数组在 DATA 步的程序数据向量（PDV）中的分配情况：

程序3-31 自定义数组下标范围不影响PDV变量命名

```
data mydata;
  array a [2:4] $ 4 ('ABCD','BCDE','CDEF');
  put _all_;
run;
```

系统输出：a1=ABCD a2=BCDE a3=CDEF _ERROR_=0 _N_=1

如果用户希望像 C/C++ 那样使用数组，可自定义下标的起止（见程序 3-32），将起始索引改成 0 的同时别忘了遍历时需要提供正确的下标，其最大下标应该是数组长度 DIM（MYARRAY）-1 而不再是 DIM（MYARRAY）。

程序3-32 自定义数组下标的访问

```
data mydata;
  array myarray [0:2] $  ('ABCD','BCDE','CDEF');
  do i=0 to dim(myarray)-1; /*自定义下标区间 0 到 dim(myarray)-1 */
    put myarray[i];
  end;
run;
```

3.3.4 隐式下标变量

在定义数组长度时在括号中除使用常量、常量区间或星号 * 外，用户也可以使用小

括号 ( ) 方式指定隐式下标变量。此时对该数组名称的访问会自动根据该下标变量的取值返回数组中特定位置的数组元素。也只有当数组定义时指定了隐式下标变量时，数组才可使用 DO-OVER 循环语句进行数组遍历。隐式下标变量必须使用小括号（）指定，不能使用中括号。而且指定了隐式数组下标的数组，普通的显式数组下标访问（比如 a[1]）不可再用。其定义语法如下：

```
ARRAY 数组名称 (隐式下标变量名) <$> <元素长度> 变量列表 <(初始值)>;
```

考察如下代码（见程序 3-33）：

程序3-33　隐式下标是指向数组元素的索引

```
data _null_;
  /*必须使用小括号指定 i 为隐式下标变量*/
  array myarray (i)    $ x1-x3 ('ABCD','BCDE','CDEF');
  *put myarray[1]=; /*注意：显式下标索引不再可用*/

  i=2; /*只能使用该变量 i 作为数组的索引，此时访问数组就是隐式下标所指向的数组元素*/
  put i myarray;

  do i=1 to 3;
    put i myarray @@;
  end;
  put;
  do over myarray; /*DO-OVER 循环进行访问*/
    put i myarray @@;
  end;
run;
```

系统输出：

```
2 BCDE
1 ABCD 2 BCDE 3 CDEF
1 ABCD 2 BCDE 3 CDEF
```

这种隐式数组下标访问方式在很多语言中已经消失，SAS 保留它是为了向下兼容，以及在需要浮动指针引用某个变量集时才使用。

3.3.5　多维数组

如果数组有 2 个以上的维度，则称之为多维数组，数组的维度之间使用逗号分隔，但初始值的指定依然用线性方式指定。其基本语法形式如下：

```
ARRAY myarray [数组长度1, 数组长度2];
ARRAY myarray [数组长度1, 数组长度2] x1-x4 y4-y4;
```

下面的例子展示了 SAS 中二维数组的定义和使用（见程序 3-34）：

程序3-34　二维数组示例

```
data mydata;
  array myarray [2, 4] (1000, 2000, 3000, 4000,
                        1500, 2500, 3500, 4500);
  i=1;
```

```
  do while (i<=2);
    j=1;
    do while(j<=4);
      put "[" i ", " j "]=" myarray[i,j] @@;
      j=j+1;
    end;
    put ;
    i=i+1;
  end;
  drop i j;
run;
proc print data=mydata;
run;
```

系统输出：

```
[1 , 1 ]=1000 [1 , 2 ]=2000 [1 , 3 ]=3000 [1 , 4 ]=4000
[2 , 1 ]=1500 [2 , 2 ]=2500 [2 , 3 ]=3500 [2 , 4 ]=4500
```

实际上，尽管SAS提供了多维数组引用方式，但在DATA步的PDV中，SAS依然是用一维变量进行存储的。即使使用PROC PRINT，系统也只会打印出构成数组的变量名，而不是二维排列的数组元素，如图3-4所示。

Obs	myarray1	myarray2	myarray3	myarray4	myarray5	myarray6	myarray7	myarray8
1	1000	2000	3000	4000	1500	2500	3500	4500

图3-4 数组在PDV中的真实存储

3.3.6 临时数组

如果在定义数组时使用关键字_temporary_进行修饰，则该数组为临时数组（见程序3-35）。临时数组的“临时”是指数组不需要映射变量列表，适用于存储计算过程的中间结果。临时数组不能被输出到目标SAS数据集，也不能用_all_打印出来。临时数组和非临时数组一样，只在当前DATA步执行期间有效，且不能跨数据步使用。

程序3-35 临时数组示例

```
data _null_;
  array array_n [3]   _temporary_ (0.05 0.08 0.12);
  array array_c [3] $ _temporary_ ('ABCD' 'BCDE' 'CDEF');

  put array_n[2] array_c[2] _all_ ;
run;
```

系统输出：

```
0.08 BCDE _ERROR_=0 _N_=1
```

临时数组与非临时数组行为几乎完全相同，但临时数组元素不能被输出到SAS数据集，且在DATA步循环时不会被重置为缺失值，因而具有自动保留特性。临时数组元素没有对应的变量映射，只能通过临时数组名和下标进行引用。临时数组定义时还必须显示指定数组元素的个数或者索引区间，且不能使用*号指定数组长度。

临时数组具有广泛的用途，其中之一是可用来简化多分支结构，如下面的代码（见程序3-36）根据4个level分别使用不同的利率进行计算。

程序3-36 多分支逻辑简化前

```
data _null_;
  amount=100;

  input level @@;
  if      level eq 1 then amount = amount * (1 + 0.05);
  else if level eq 2 then amount = amount * (1 + 0.10);
  else if level eq 3 then amount = amount * (1 + 0.15);
  else if level eq 4 then amount = amount * (1 + 0.20);

  put level amount "," @@;
  datalines;
1 2 3 4
run;
```

上面的4个分支语句可以通过如下方式定义一个利率表，然后自动根据利率表进行计算（见程序3-37）。

程序3-37 利用临时数组简化多分支逻辑

```
data _null_;
  amount=100;

  input level @@;
  array rate [4] _temporary_ (0.05 0.10 0.15 0.20);
  amount = amount *  (1 + rate[level] );

  put level amount "," @@;
  datalines;
1 2 3 4
run;
```

系统输出：1 105, 2 110, 3 115, 4 120

3.3.7 数组排序

SAS提供系统例程SORTN、SORTC对数组进行排序，其中SORTN用于数值型变量的排序，SORTC用于字符型变量排序。在SAS中例程（Routine）就是一个没有返回值的子函数，调用时须用类似CALL SORTC(...)的格式。考察如下代码（见程序3-38），你会发现一旦调用排序代码，数组中的值会按照指定顺序重排，从而引用同样位置的数组元素将获得不同的结果值。

程序3-38 数值数组和字符串数组的排序

```
data _null_;
  array array_n{3} (0.3 0.1 0.2);
  put "Before:  " array_n[*] ;
  call sortn( of array_n[*] );/*对数值数组进行排序*/
  put " After:  " array_n[*] ;

  array array_c{3} $ y1-y3 ("BCDE" "ABCD" "CDEF") ;
  put "Before:  " array_c[*] ;
```

```
  call sortc( of y1-y3);/*对字符数组进行排序*/
  put " After:  " array c[*] ;
run;
```

系统输出：

```
Before:  0.3 0.1 0.2
 After:  0.1 0.2 0.3
Before:  BCDE ABCD CDEF
 After:  ABCD BCDE CDEF
```

3.3.8　注意事项

SAS 数组与其他计算机语言不同，程序员在编程实践中很容易发生如下几种类型的错误，是 SAS 数组的独特行为所致。

（1）引用未声明的数组错误：在一个数据步中试图对另一个数据步内定义的数组执行运算时，系统一般会报告如下编译错误。

ERROR：引用了未声明的数组：y。
ERROR：变量 y 没有声明为数组。

因为 SAS 编译执行是以步（Step）为单位，所以数组作为变量的组织形式不能够跨多个数据步。如果需要在多个步之间调用，可使用 SAS 宏变量或中间临时数据集在步（Step）之间进行传递。用户也可以将不同步的代码合并进行分析，比如下面的例子（见程序 3-39）：

程序3-39　企图跨数据步访问数组

```
data _null_;
  array y [3] (100,200,300);

  do i=1 to 3;
    put  y[i] @@;
  end;
  drop i;
run;
data _null_;/*如下代码试图对数组 y 进行求和*/
  sum=0;
  do i=1 to 3;
    sum=sum+ y[i];
  end;
  drop i;
run;
```

第一种解决方法是先用数据集 mydata 传递数值，然后用 SET 语句对输入数据叠加运算逻辑来实现（见程序 3-40），输出结果如图 3-5 所示。

程序3-40　使用数据集作为桥梁实现跨数据步访问

```
data mydata;
  array y [3] (100,200,300);

  do i=1 to 3;
    put y[i] @@;
  end;
```

```
  drop i;
run;

data mydata;
  set mydata; /*利用数据集做桥梁*/
  array y [3];

  sum=0;
  do i=1 to 3;
    sum=sum+ y[i];
  end;
  drop i;
run;
```

Obs	y1	y2	y3	sum
1	100	200	300	600

图 3-5　输出数据集

第二种解决方法是，如果可能，则把数组的运算逻辑封装在单一数据步内（见程序 3-41）。

程序3-41　运算逻辑封装在单个数据步内

```
data mydata;
  array y [3] (100,200,300);

  do i=1 to 3;
    put  y[i] @@;
  end;

  sum=0; /*运算逻辑二*/
  do i=1 to 3;
  sum=sum+ y[i];
  end;
  drop i;
run;
```

（2）数组下标越界错误：当数组下标不在数组定义的合法下标范围内时引发，包括低于数组下界或者高于数组上界；有时候是因为使用 DO 循环条件不当，或因错误自定义了数组下标范围。比如程序 3-42 只有 3 个元素却用 1~4 去访问，或者定义了下标区间 2:4，结果却用 1~3 去访问。

程序3-42　数组下标越界示例

```
data mydata;
  array y [3] (100,200,300);

  i = 1;
  do while (i <=4); /*或 do until (i >4); */
    put "y[ " i "]= " y[i] ; /*最后一个出错*/
    i=i+1;
  end;
  drop i;
run;

data mydata;
```

```
  array y [2: 4] (100,200,300);
  i = 1;
  do while (i <=4);
    put "y[ " i "]= " y[i] ; /*第一个出错*/
    i=i+1;
  end;
  drop i;
run;
```

（3）指定的数组下标过多错误：当数组名与函数名冲突时，SAS会将该名称之后的括号当作数组引用而非函数引用进行处理，此时本应该作为函数参数进行传递的变量，被当作数组下标进行处理，就会引发各种错误信息。修正办法就是正确对数组进行命名，避免与系统函数或其他自定义用户函数重名。下面的例子（见程序3-43）本意是调用mean函数计算数组元素的均值。

程序3-43　数组名与函数名冲突引发伪“数组下标过多错误”

```
data mydata;
  array mean [3]  (100,200, 300);
  avg= mean(of mean1-mean3) ;/* ERROR：为数组"mean"指定的数组下标过多*/
  put avg=;
run;
```

正确的做法是改变用户数组mean的名称，避免与系统函数mean重名。

（4）把数组元素完全当变量使用：由于ARRAY语句是DATA步内的编译语句，数组元素的引用方式不能完全等同于变量，比如在DROP、KEEP等语句中，用数组元素引用方式会引发编译语法错误。变通的方法是使用该数组元素对应的变量名。比如下面的例子（见程序3-44）若用DROP语句将第一个变量从输出变量列表中删除，就会引发语法错误。正确的做法是用PDV中的具体变量y1。

程序3-44　数组元素完全当作变量使用：数组只是组织方式，只有PDV变量才是真的变量

```
data mydata;
  array y [3]  (100,200,300);
  drop y[1]; /*应该使用 drop y1; */
run;
```

数组操作是很多复杂数据分析算法的基础，利用SAS数组可以将传统编程语言中的各种算法在SAS实现，从而完成自定义分析算法的编程工作。SAS程序实现大规模数据处理可以基于SAS数组、SAS数据集以及专门面向矩阵运算的SAS/IML。一般情况下，涉及大量重复运算时不要基于数据集，而应该使用SAS数组这种存在于PDV内存空间的语言元素来实现；其根本原因是SAS数据集操作涉及数据持久化，即需要大量的磁盘I/O读写操作而严重影响系统性能。这是很多学统计出身的数据分析人员容易犯的错误。

第4章

流程控制

计算机编程语言经过几十年的演化，语法元素基本上都很成熟。所谓语法本质上就是关键字、符号以及它们如何组合的规范。各种编程语言在数据类型与结构、流程控制、实体引用与代码重用、设计哲学等方面大不相同，但它们在计算机编程的语义上大体相同。用户如果洞悉了某种计算机语言的设计精髓，则精通多种计算机语言并非难事。各种计算机语言都有一个重要的部分也就是执行流程控制，本章将主要讲述 SAS 语言的流程控制有关内容。

所有的编程语言都支持代码自上而下的顺序处理，这与人类的阅读习惯和文件处理顺序是一致的。在逻辑上，顺序处理、分支控制、循环控制是实现计算机流程控制的三大核心构件，任何复杂的计算逻辑都可以由这 3 种方式组合而成。现代计算机语言大都会提供强大的实体引用和代码重用技术，如静态代码包含、宏替换、函数封装、面向对象、代码模板以及泛型等，从而使分解问题、解决问题编写大规模复杂程序成为可能。顺序、分支和循环作为构成程序流程控制的三大部分，其中顺序执行是自然隐含的行为，因此本章重点探讨 SAS 的分支控制和循环控制。

4.1 DO-END 语句块

在讲述流程控制之前有必要先讲 DO-END 语句块，在 SAS 语言中，我们可利用 DO 和 END 两条语句将包含在它们之间的多条语句“封装”为单个语句块，DO-END 语句块在语法上被视为单条语句进行处理。因此它类似于 C/C++ 和 Java 语言中的 {} 代码分组符。这种分组语句最常见的用法是在分支控制和循环控制中用来实现语句块功能和代码嵌套，但 DO-END 语句块可用在 SAS 数据步内任何地方用于控制 SAS 代码的分组和隔离。其基本语法为：

```
DO;
  Statement-1;
  ...
  Statement-n;
END;
```

DO-END 语句块的简单演示如程序 4-1 所示。

程序4-1 DO-END语句块示例

```
data  null ;
  n 4;
  do;
    put "Hello World";
    n=n+2;
  end;

  if mod(n, 2)=0 then
  do;
    sum+n;
    put n ;
  end;
run;
```

4.2 分支控制

4.2.1 IF-THEN 分支控制

分支控制用于构造运行时可能的执行路径。如果表达式为真，则执行条件为真的语句或语句块，否则执行条件为假对应的语句或语句块，其中条件为假的部分是可选的。其基本语法如下：

```
IF exp THEN True-Statement;
<ELSE False-Statement;>
```

在 SAS 语言中，表达式求值结果可能为缺失值、零或者非零值。 SAS 视缺失值和零为假，非零值作为真。分支控制中 ELSE 语句并不是必须的，也就是说可以仅在表达式为真时作某种处理，条件不成立时可以不处理。下面的例子 (见程序 4-2) 首先生成一个满足正态分布的随机数，如果该随机数大于等于 0，则设置 isPositive 为 1，表示它是一个正数。

程序4-2 生成一个正态分布随机数，IF-THEN检测它是否为正数

```
data _null_;
  x=rand('NORMAL');

  isPositive=0;
  if x >=0 then isPositive=1;

  put x= isPositive=;
run;
```

上面的例子也可以改写为包含 ELSE 语句，表示小于等于 0 的时候将 ispositive 设置为假。这两个例子的逻辑是等价的（见程序 4-3）。

程序4-3 IF-THEN-ELSE 双分支控制

```
data  null ;
  x rand('NORMAL');
```

```
  if x > 0 then isPositive=1;
  else isPositive=0;

  put x= isPositive=;
run;
```

输出：x=-0.518828349 isPositive=0

如果在 IF-THEN 的处理中需要执行多条语句，我们可以使用前面讲到的 DO-END 语句块功能。DO-END 语句块中可包含任何其他 SAS 语句，或者嵌套别的 DO-END 语句块，或者嵌套其他分支控制语句和循环控制语句组成的语句块。当 IF-THEN 语句扩展到 DO-END 语句块时，其基本语法形式就为：

```
IF exp THEN True-Statement;
<ELSE False-Statement;>
```

演变为如下形式，即 *True-Statement* 和 *False-Statement* 都变为 DO-END 语句块。

```
IF exp THEN
  DO;
    True-Statement-1;
    ...
    True-Statement-n;
  END;
<ELSE
  DO;
    False-Statement-1;
    ...;
    False-Statement-n;
  END;>
```

如下代码利用 SAS 强大的随机数生成器模拟一次抛硬币的过程，抛硬币是单次试验成功（成功：国徽朝上；失败：国徽朝下）发生概率为 0.5 的伯努利试验，它服从只有两个可能取值的等概率均匀分布（见程序 4-4）。

程序4-4 模拟伯努利试验

```
data _null_;
  x=rand('BERNOULI', 0.5);

  if x=1 then do;
    isUp=1;
    put "硬币国徽朝上 " isUp=;
  end;
  else do;
    isUp=0;
    put "硬币国徽朝下 " isUp=;
  end;
run;
```

DATA 步支持一个很特殊的 IF **取子集**语句，它表示在 IF 条件满足时才继续往下处理，否则返回 DATA 的开始处。这个语句没有 IF-THEN 语法结构而只有 IF 表达式，因此千万不要将它与这探讨的 IF-THEN 条件分支控制语句混为一谈。IF **取子集**语句的语法为 IF *EXPRESSION*; 如程序 4-5 所示：

```
程序4-5  IF-取子集语句并非IF-THEN分支控制
data myclass;
  set sashelp.class;
  if age>13;/*仅当 sashelp.class 中 age>13 才继续执行*/
run;
```

4.2.2 ELSE-IF 多分支控制

当我们需要多个分支控制时，可以使用多条ELSE IF语句，在语法等价于在IF和ELSE语句之间插入任意多个ELSE IF语句或语句块，从而实现多分支控制。其基本语法如下：

```
IF exp-1 THEN True-Statement;
<ELSE IF exp-2 THEN Statement2; >
<ELSE IF exp-n THEN Statementn; >
ELSE False-Statement;
```

实际上，这种形式的多分支控制等价于在ELSE语句中使用嵌套的DO-END语句和IF-THEN语句实现，只不过是将配对的DO-END语句简化而已；上面的代码等价于如下代码：

```
IF exp-1 THEN True-Staement;
ELSE
DO;
  IF exp-2 THEN Statement-2;
  ELSE
  DO;
    IF exp-n THEN Statement-n;
    ELSE False-Statement;
  END;
END;
```

4.2.3 SELECT-WHEN 多分支控制

SAS提供类似于C/C++和JAVA语言的SWITCH-CASE多分支控制语法，且各分支语句默认是相互隔离（即自带break功能）。其基本语法如下：

```
SELECT <(exp-0)>;
  WHEN (exp-1) Statement-1;
  ..
  WHEN (exp-n) Statement-n;
  <OTHERWISE Statement-x;>
END;
```

SAS首先对SELECT语句中的表达*exp-0*求值，然后依次尝试匹配WHEN语句后的表达式（如*exp-1*），如果两者求值匹配则执行对应WHEN语句后的语句（如*Statement-1*）并跳出多分支控制。在C/C++和JAVA语言中，SWITCH-CASE分支语句是依次顺序执行，只有遇到人为添加的break语句才会跳出多分支控制块。SAS的SELECT-WHEN多分支控制语句相当于自带break功能。

下面的例子（见程序4-6）模拟抛一次麻将骰子的过程，骰子的点数服从离散均匀分布，即骰子6个面上的数字被抛到的概率是均等的，都是1/6。

程序4-6 SELECT-WHEN多分支控制

```
data  null ;
  x= ceil(rand("UNIFORM")* 6);
  put "x=" x;

  select(x) ;
    when (1) put "I";
    when (2) put "Ⅱ";
    when (3) put "ⅡI";
    when (4) put "IV";
    when (5) put "V";
    when (6) put "VI";
    otherwise put "***";
  end;
run;
```

如果在SELECT语句的括号中指定了表达式x，则WHEN中也必须是一个或多个表达式，只要遇到SELECT语句和WHEN语句中的表达式求值相等即执行该分支并跳出分支控制。然而，SELECT语句也可以不指定表达式，此时分支的执行完全依赖于WHEN语句后指定的表达式求值是否为真，它可用来实现更加灵活的多分支控制结构（见程序4-7）。

程序4-7 SELECT-WHEN 多分支控制的另一种写法

```
data _null_;
  x= ceil(rand("UNIFORM")* 6);
  put "x=" x @;

  select;/*千万不要写成 select(x);*/
    when (x=1, x=2, x=3) put "SMALl";
    when (x=4 or x=5 or x=6) put "BIG";
    otherwise put "IMPOSSIBLE";
  end;
run;
```

4.3 循环控制

SAS语言有强大的循环控制支持，包括指定次数的循环（即C/C++语言中的FOR循环）、指定条件的循环（DO-WHILE和DO-UNTIL）、指定集合的循环（即某些高级语言的FOR EACH循环）三大类。在SAS语言中它们都统一由不同形式的DO语句实现。

4.3.1 指定次数的循环：DO-TO-BY

当循环次数固定时，可以使用定数循环，如在遍历数据集或者数组元素时比较常用。其中<BY *Step*>部分是可选的，循环步长*Step*缺省为1。当我们逆向循环时需要指定*Step*为-1。其基本语法为

```
DO var Start-Value TO End Value <BY Step>;
  Statement or Statement Block;
END;
```

比如对于一个一维数组，我们可以用循环语句将每一个元素打印出来，其中 DIM 函数用于返回数组的长度（见程序 4-8）。

程序4-8 一维数组的遍历

```
data _null_;
  array myarray [5] _temporary_ (1,2,3,4,5);
  do i=1 to dim(myarray) by 2; /*打印 1, 3, 5 */
    put myarray[i] @@;
  end;
run;
```

系统输出如下：1 3 5

4.3.2 指定条件的循环：DO-WHILE 与 DO-UNTIL

DO-WHILE 循环：当特定条件为真的时候执行循环体中的语句，循环体执行结束后再次判断循环条件是否为真，如果为真则继续执行体，否则结束循环。其基本语法为

```
DO WHILE (exp);
  Statement or Statement Block;
END;
```

比如 1 到 100 求和，即可使用如下方法进行（见程序 4-9）。

程序4-9 DO-WHILE示例

```
data _null_;
  sum=0;
  n=1;
  do while(n<=100);
    sum=sum+n;
    put n= sum=;
    n=n+1;
  end;
run;
```

输出结果如下：n=1 sum=1…n=100 sum=5050

DO-UNTIL 循环：首先执行循环体，直到表达式为真的时候结束循环。DO-UNTIL 循环控制语句被用于至少需要执行一次循环体的情况。其基本语法如下：

```
DO UNTIL (exp);
  Statement or Statement Block;
END;
```

比如 DO-UNTIL 的循环为 n 大于 100，这时当 n 等于 100 时也会执行循环体。程序 4-10 执行结果与程序 4-9 的输出完全等价。

程序4-10 DO-UNTIL示例

```
data _null_;
  sum=0;
  n=1;
  do until(n>100);
    sum=sum+n;
    put n= sum=;
    n=n+1;
  end;
run;
```

理论上，任何DO-UNTIL循环都可以通过改变判断条件用DO-WHILE循环实现。实际上，SAS语言极其灵活，用户甚至可以将DO-TO-BY语句和DO-WHILE/DO-UNTIL结合使用，这种灵活机制在函数封装中可补偿由于SAS94M3及更早版本的PROC FCMP不支持LEAVE语句造成的缺憾。考察如下两段代码（见程序4-11），它们实现了相同的循环控制逻辑。

程序4-11 DO-TO-BY结合DO-WHILE/DO-UNTIL 语句

```
data _null_;
  do i=1 to 100 ;
    if i=50 then leave;
  end;
  put i=; /*输出 i=50*/
run;

data _null_;
  do i=1 to 100 until (end =1);
    if i=50 then end=1;
  end;
  put i=;
run;
```

4.3.3 指定集合的循环：DO-OVER

在编程中需要大量处理集合类型的数据，如数组、队列和字典等。Java/C# 语言都提供 for each 的循环来遍历集合元素，其语法结构为

for (type element : set) { statements } 和 **foreach**(type element **in** set) { *statements* }。

在SAS语言中则提供一个特殊的DO-OVER语句，可用来对用小括号指定了隐式下标变量的数组进行元素遍历（见程序4-12）。

程序4-12 DO-OVER循环示例

```
data _null_;
  array myarray(i) $ x1-x4 ('The','Power','To', "Know");

  length s $32767;
  s="";
  do over myarray;
    put i "-" myarray @@;
    s = trim(s) || " " || myarray;
  end;
  put;
```

```
  put "Merge: " s;
run;
```

系统输出结果为

```
1 -The 2 -Power 3 -To 4 -Know
Merge: The Power To Know
```

4.4 特殊的流程控制语句

在 C/C++ 和 Java 语言中，我们可以用 break 语句或 continue 语句在适当条件下结束循环或忽略后续语句直接进入下一次循环的条件判断。通常 break 或多分支控制块，可用于直接结束循环，而 continue 语句表示不再执行循环体中 continue 语句之后的那些语句，直接跳到下一次循环开始的条件判断处。这些语句都是用来控制执行流程的特殊语句。

4.4.1 跳出循环语句：LEAVE

SAS 程序中使用 LEAVE 语句来离开循环，等价于 C++/Java 中的 break 语句。LEAVE 语句可用于 DO 循环块和 SELECT 多分支语句块（见程序 4-13）。

程序4-13 跳出循环LEAVE

```
data _null_;
  array myarray [5] _temporary_ (1,2,3,4,5);

  do i=1 to dim(myarray);
    if (i=3) then leave;
    put myarray[i]=;
  end;
run;
```

输出：因为当进行到第 3 次循环时跳出循环体，所以它不输出 myarray[3]=3 的情况。

4.4.2 继续循环语句：CONTINUE

SAS 程序中使用 CONTINUE 语句来忽略循环体中该语句之后的所有语句，跳到循环开始处的条件判断继续循环。它等价于 C++/JAVA 中的 CONTINUE。CONTINUE 语句只能用于 DO 循环中（见程序 4-14）。

程序4-14 Continue 语句

```
data _null_;
  array myarray [5] _temporary_ (1,2,3,4,5);

  do i=1 to dim(myarray);
  if (i=3) then continue; /*直接跳到 do 继续下一次循环处*/
    put myarray[i]= @@;
```

```
  end;
run;
```

上面的代码输出如下，其中 i-3 被忽略。

```
myarray[1]=1 myarray[2]=2 myarray[4]=4 myarray[5]=5
```

4.4.3　返回语句：RETURN

SAS 数据步的 RETURN 语句表示在当前点上立即停止执行，返回本 DATA 步的开始处继续执行，此时数据步会将当前观测从 PDV 自动输出到目标数据集中。但如果当前 RETURN 语句是从 LINK 语句跳入执行的，则 RETURN 语句将返回该 LINK 语句的下一条 SAS 语句处继续执行。实际上，SAS 数据步的最后一个语句执行后，系统会默认运行 RETURN 语句到数据步的开始处继续下一次隐形循环。下面的例子（见程序 4-15）将过滤掉输入数据中两个变量差值的绝对值大于 10 的数据。

程序4-15　RETURN语句

```
data _null_;
  input x y @@;
  if abs(y-x)>10 then return;

  put x= y=;
  datalines;
10 20
20 35
30 40
40 25
50 60
;
```

程序 4-15 输出如下所示：

```
x=10 y=20
x=30 y=40
x=50 y=60
```

在 PROC FCMP 和 PROC DS2 的函数封装技术中，RETURN 语句跟其他计算语言的语义类似，用来从函数封装中设置返回值。而本节探讨的是 DATA 步中的 RETURN 语句的行为。

4.4.4　中止执行语句：STOP 与 ABORT

在 DATA 步中，STOP 语句可以用来正常停止当前数据步的执行，并马上处理下一个 DATA 步或 PROC 步。此时，执行流程立即结束，PDV 中已经处理完毕的观测也不会被输出到目标数据集中，STOP 语句可出现在 DATA 步中的任何地方。

如果 SAS 运行在 Windows 平台上，ABORT 语句也可以用来停止当前 DATA 步的执行，区别是 ABORT 语句会设置 DATA 步的系统变量_ERROR_为 1，而 STOP 语句则不会设置错误标志。也就是说，ABORT 相当于人为结束程序并引发错误在系统日志写入

错误信息，这种中止通常用于检测到异常时结束当前DATA步的执行，比如：

```
ERROR: ABORT 语句终止了执行，位置： 行 5 列 23。
```

另外，当SAS在批处理/非交互模式下运行时，ABORT和STOP也有不同的行为，如果要继续处理后续的DATA步和PROC步，需要使用STOP语句。最常见的一个用途是把STOP语句放在程序的最后面，以防止对某个外部数据的随机访问发生挂起。比如下面的程序（见程序4-16）用于从系统数据集sashelp.class中等距抽样4条数据到myclass中，规则是每5行抽取1行，输出结果如图4-1所示。

程序4-16 随机访问数据的最后需要STOP 语句防止发生死循环。

```
data myclass;
  do i=1 to count by 5;
    set sashelp.class point=i nobs=count;
    output;
  end;
  stop;
run;
proc print; run;
```

执行上面的代码将对SASHELP.CLASS抽样产生一个小样本（见图4-1右表）。此时如果没有STOP语句可能导致程序不返回的情况发生。

Obs	Name	Sex	Age	Height	Weight
1	阿尔弗雷德	男	14	69.0	112.5
2	爱丽丝	女	13	56.5	84.0
3	芭芭拉	女	13	65.3	98.0
4	凯露	女	14	62.8	102.5
5	亨利	男	14	63.5	102.5
6	詹姆斯	男	12	57.3	83.0
7	简	女	12	59.8	84.5
8	简妮特	女	15	62.5	112.5
9	杰弗瑞	男	13	62.5	84.0
10	约翰	男	12	59.0	99.5
11	乔伊斯	女	11	51.3	50.5
12	茱迪	女	14	64.3	90.0
13	罗伊斯	女	12	56.3	77.0
14	玛丽	女	15	66.5	112.0
15	菲利普	男	16	72.0	150.0
16	罗伯特	男	12	64.8	128.0
17	罗纳德	男	15	67.0	133.0
18	托马斯	男	11	57.5	85.0
19	威廉	男	15	66.5	112.0

Obs	Name	Sex	Age	Height	Weight
1	阿尔弗雷德	男	14	69.0	112.5
2	詹姆斯	男	12	57.3	83.0
3	乔伊斯	女	11	51.3	50.5
4	罗伯特	男	12	64.8	128.0

图4-1 随机访问实现等距抽样

4.4.5 跳转语句：GOTO与LINK

虽然很多现代编程语言已经抛弃了GOTO语句，但我们不能否认GOTO语句是个万能的恶魔。GOTO语句可以让SAS程序执行立即跳到本DATA步内的指定标签处。

在任何一个SAS语句前面都可定义一个标签，表示特定语句的执行入口。如果跳转的标签后是一个RETURN语句，则执行将返回DATA步的开始处。在SAS语言中，GOTO也并非毫无原则，它只允许在同一DATA步的程序空间内跳转，究其原因是SAS代码以步为单元编译执行。GOTO语句的基本语法为

```
GO TO label;
```

SAS的GOTO语句中间是带空格的，但推荐使用它不带空格的别名形式GOTO，与别的计算机语言一样。

```
GOTO label;
```

考察如下代码的执行（见程序4-17）。

程序4-17 万能的GOTO在DATA步内自由跳转

```
data _null_;
  array myarray [5] _temporary_ (1,2,3,4,5);
  do i=1 to dim(myarray);
    put myarray[i]=;
    goto mylabel;
  end;
mylabel:
    put "Go to here";
run;
```

系统输出如下：

```
myarray[1]=1
Go to here
```

程序4-17说明GOTO语句从循环体中跳转到标签MYLABEL处，随后结束当前DATA步的执行。理论上GOTO语句可实现任意流程控制，如下面的代码（见程序4-18）用IF语句和GOTO语句实现循环控制结构：

程序4-18 IF-THEN 和 GOTO语句实现循环控制结构

```
data _null_;
  i=1;
loop:
  put i= @@;
  i=i+1;
  if i<5 then goto loop;
run;
```

系统输出为 i=1 i=2 i=3 i=4

SAS语言中还有一个比无条件跳转语句GOTO弱一点的流程跳转语句LINK，它的功能类似GOTO，两者的区别是后面遇到RETURN语句的行为有所不同。SAS数据步的LINK语句通常用于实现扁平模式下的函数封装，对此将在函数封装一节详细讲述。

SAS语言不但提供现代编程语言都有的流程控制机制，同时也保留了很多从SAS

语言定义之初就有的语言元素。应用这些流程控制机制，程序员就可灵活地写出各种各样数据处理程序。通过数据分析行业几十年的实践证明，完成同样数据处理功能的 SAS 程序远比 Java C++ 程序简洁而高效。况且连万能的 GOTO 语句至今 SAS 都保留着，还有什么程序控制是不能实现的？当然，程序除了流程，还需要代码封装和复用，对这些将在后续章节进行讲述。

第5章

函数封装

计算机编程语言用来告诉计算机做什么以及如何做。编程思想的发展史就是人类不断总结如何控制机器，实现丰富多彩程序世界的历史。在计算机发展的早期阶段，编程技术既没有我们现在熟知的结构化思想，也没有函数封装概念，更没有面向对象和泛型；所有可用的编程元素就是现在汇编语言中大家能看到的那些指令集，那时的编程几乎就是告诉计算机 CPU 如何加载数据和指令，如何与寄存器打交道，执行以及跳转等。

后来结构化编程思想日趋成熟，面向过程的编程技术开始兴起，过程和函数封装的概念才开始流行起来。如今函数仍然是我们封装计算逻辑的基本形式。函数定义由函数名称、参数列表、基于参数和变量作用域的计算逻辑以及函数返回值构成。在计算机内存中，一个操作系统可以调度的进程在内存分配上包含代码段和数据段，数据段之上是由函数编译形成的代码堆；顶部则是堆栈，该堆栈中每一个栈帧代表一次函数调用，包括上一栈帧的地址、输入参数、返回值及返回地址等信息。虽然很多程序员已经不再探求函数编译和调用的细节，但作为 SAS 数据科学家还是需要深刻理解这一点，才能掌握 SAS 编程的精髓。首先我们回顾一下在计算机语言发展的早期是如何实现函数封装这一概念的？这可以从程序设计中万能的 GOTO 语句之命运开始说起。

GOTO 语句的作用非常简单，就是将程序执行跳转到指定语句。然而，结构化程序设计之父 Edsger Wybe Dijkstra 在 1968 年的论文“*Go To Statement Considered Harmful*”中指出：GOTO 语句会使分析和验证程序正确性（尤其是循环控制）的任务变得复杂，认为不加限制地使用 GOTO 语句应当从高级语言中废止。而另外，经典巨著 *The Art of Computer Programming* 的作者 Knuth Donald Ervin 在 1974 年的论文“*Structured Programming with go to Statement*”里分析了许多常见的编程任务，认为其中一些使用 GOTO 语句能得到最理想的程序结构，有控制地使用一些 GOTO 语句是必要的。这就是计算机编程历史上著名的 GOTO 语句之争。

我们这个时代最伟大的两位计算机科学家尚且对 GOTO 语句存在合理性的看法不同，我们就不必进一步探讨其存亡的合理性，而应该关注如何合理使用它。现实情况是 C/C++ 语言到目前为止仍然保留了 GOTO 语句，而 Java 语言虽然不支持 GOTO 语句却保留了 GOTO 关键字，C# 语言则坚定支持 GOTO 语句。尽管我们已经迈入了面向对象和泛型的高级编程时代，但 GOTO 语句在跳出多重循环，处理运行异常等资源清理方面极其简洁高效。据统计，Linux-2.6.21 内核代码中就使用了超过 20333 个 GOTO 语句。在某种意义上，GOTO 语句是计算机汇编语言时代 JMP 指令在高级语言中的残留，而高级计算机语言分支循环控制中普遍存在的 BREAK 和 CONTINUE 语句在本质上也是一

种受限的 GOTO 语句。

SAS 语言至今仍然依然保留了 GOTO 语句。下面的例子（见程序 5-1）展示 GOTO 语句的一个经典使用场景——在多层嵌套循环中跳出循环体。

程序5-1 使用GOTO 语句从多层循环中跳出

```
data _null_;
  do x=1 to 10;
    do y=1 to 10;
      do z=1 to 10;
        if (x*y*z=125) then goto exit;
      end;
    end;
  end;
  return;
exit:
  put x= y= z= 'EXIT LOOP AT x*y*z=125';
run;
```

如果不用 GOTO 语句，等价的实现代码如下（见程序 5-2），但它用了 GOTO 语句的变体 LEAVE 语句。

程序5-2 使用LEAVE 语句从多层循环中跳出

```
data _null_;
  found=0;
  do x=1 to 10;
    do y=1 to 10;
      do z=1 to 10;
        if (x*y*z=125) then found=1;
        if found=1 then leave;
      end;
      if found=1 then leave;
    end;
    if found=1 then leave;
  end;
  if found=1 then do;
    put x= y= z= 'EXIT LOOP AT x*y*z=125';
  end;
run;
```

或者使用 RETURN 语句实现跳出循环（见程序 5-3）。

程序5-3 使用RETURN 语句从多层循环中跳出

```
data _null_;
  found=0;
  do x=1 to 10;
    do y=1 to 10;
      do z=1 to 10;
        if (x*y*z=125) then do;
          put x= y= z= 'EXIT LOOP AT x*y*z=125';
          return;
        end;
      end;
    end;
  end;
run;
```

GOTO 语句是无条件地执行跳转语句，理论上它可实现任何计算逻辑结构。在 SAS 语

言DATA步内实现原生的函数封装逻辑，需要用到GOTO语句的一个变体LINK语句来实现。理解它将对我们在SAS语言中实现复杂逻辑，理解编程语言底层运行机制很有帮助。

5.1 LINK-RETURN 技术

LINK 语句是一种特殊的执行跳转语句，它可以将程序执行马上跳转到同一DATA步内所指定的标签处；如果后续执行中遇到RETURN语句，执行会跳转回该LINK语句后的下一条语句继续执行。LINK语句默认最多可嵌套10层，用户可以通过DATA语句的/STACK选项来改变嵌套的LINK语句层数。

程序5-4展示了LINK和GOTO语句的区别：左边的编译结果运行4个语句，而右边只运行2个语句，左侧的RETURN语句返回LINK语句的下一条语句，而右侧的RETURN语句返回DATA步的开始处。

程序5-4 LINK语句与GOTO 语句的对比

```
data _null_;
  put "Statement1";
  link label1;
  put "Statement2";
label1:
  put "statement3";
  return;
  put "statement4";
run;
```

```
data _null_;
  put "Statement1";
  goto label1;
  put "Statement2";
label1:
  put "statement3";
  return;
  put "statement4";
run;
```

LINK语句与GOTO语句的区别是后面RETURN语句的行为（见图5-1）。

LINK *label;*

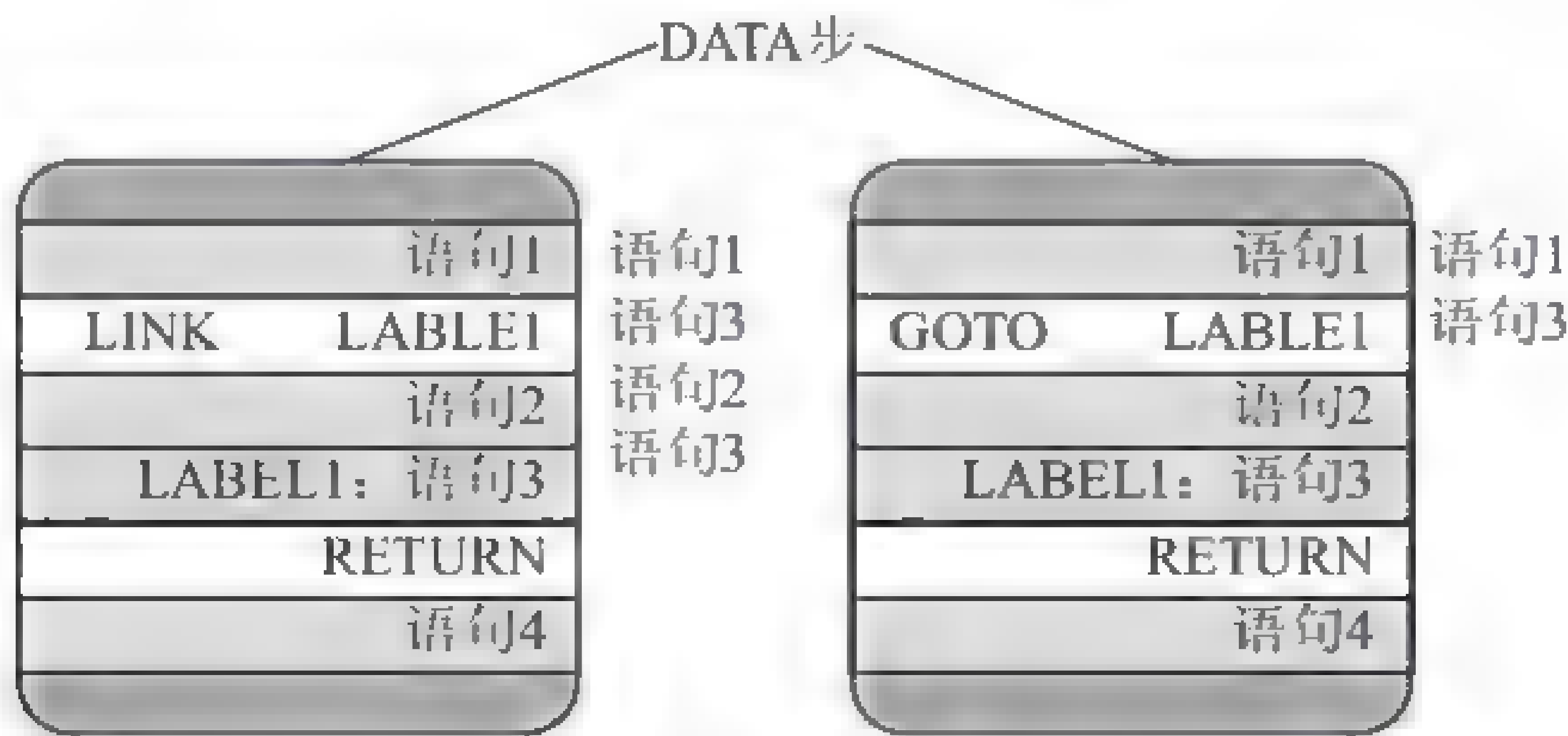


图5-1 LINK语句与GOTO语句

当LINK语句跳转执行后遇到一个RETURN语句时，执行会跳转到编译结果中该LINK语句的下一条语句或语句块继续执行。而GOTO语句跳转后遇到的RETURN语句则返回DATA步的开始处。如果GOTO语句后面又有LINK语句，则该LINK后的第一个RETURN语句会跳转到该LINK语句的下一条语句，再后面遇到的RETURN语句才会跳转到DATA步的开始处。LINK语句通常跟着显式的RETURN，而使用GOTO语句时通常不需要一个

RETURN 语句。当然，程序员有义务控制执行逻辑以防止死循环。其基本语法如下所述。

下面举一个完整的例子来说明如何使用 LINK-RETURN 实现 DATA 步原生风格的函数封装功能，通常这种技巧被称为伪函数封装技术。下面的代码（见图 5-2）实现了加法和减法功能，结果放入变量 z 。

程序5-5 LINK 语句实现伪函数封装技术

```
data test;
  x=1; y=2;
  link func_add;    /*调用函数 func_add*/
  output;           /*将运算过程 x y z 都输出到数据集 test 中 */
  put x= y= z=;     /*打印结果到日志中*/

  x=3; y=4;
  link func_sub;    /*调用函数 func_sub*/
  output;
  put x= y= z=;

  return;           /*返回 DATA 步的开始处，防止func_add 被自动执行*/

func_add:
  z=x+y;            /*执行加法运算 z=x+y*/
  return;           /*返回跳入此处的那个 LINK 语句的下一条语句*/

func_sub:
  z=x-y;            /*执行减法运算 z=x-y*/
  return;
run;
proc print data=test; run;
```

输出结果如图 5-2 所示。

Obs	x	y	z
1	1	2	3
2	3	4	-1

图 5-2　加法与减法输出结果

对一系列数据输入执行加法运算，也可以用同样的原理实现。演示代码如程序 5-6 所示：

程序5-6 LINK 语句结合INPUT 语句实现连续调用

```
data test;
  input x y;

  LINK func_add;
  output;
  put x= y= z=;
  return; /*返回 DATA 步的开始处*/

func_add:
  z=x+y;
  return; /*返回跳入此处的那个 LINK 语句的下一条语句*/
cards;
```

```
1 2
3 4
;
run;
proc print data-test; run;
```

系统输出如图 5-3 所示。

Obs	x	y	z
1	1	2	3
2	3	4	7

图 5-3 加法运算结果

如果要对这种 LINK-RETURN 封装的伪函数实现递归调用功能，需要将形式参数和实际参数分离，并且要在 DATA 步中实现“堆栈”数据结构，才能最终实现这一机制（参见第 13 章内容）。LINK-RETURN 封装技术理论上可实现非常复杂的执行控制，但要求有较高的编程技巧。

5.2 SAS 宏函数封装

SAS 语言中 SAS 宏也可以用来封装函数。实际上很多时候 SAS 程序员已经在滥用这种宏函数封装技巧。考察如下 SAS 代码（见程序 5-7），看如何在 DATA 步中将计算逻辑 z=x+y 以 SAS 宏函数方式实现。

程序5-7 计算逻辑封装之前

```
data mydata;
  input x y;
  z=x + y; /*待封装的计算逻辑*/
cards;
1 2
3 4
;
run;
proc print data=mydata;run;
```

首先利用 SAS 宏将计算逻辑 z=x+y 进行剥离并封装，这样就可以在多个数据集中重复调用该宏函数（见程序 5-8）。

程序5-8 计算逻辑封装为 SAS宏

```
%macro func_add( arg1, arg2, arg3);
  &arg3= &arg1 + &arg2;
%mend;

data mydata;
  input x y;
```

```
  %func_add(x,y,z);
cards;
1 2
3 4
;
run;
proc print; run;
```

程序输出结果如图 5-4 所示。

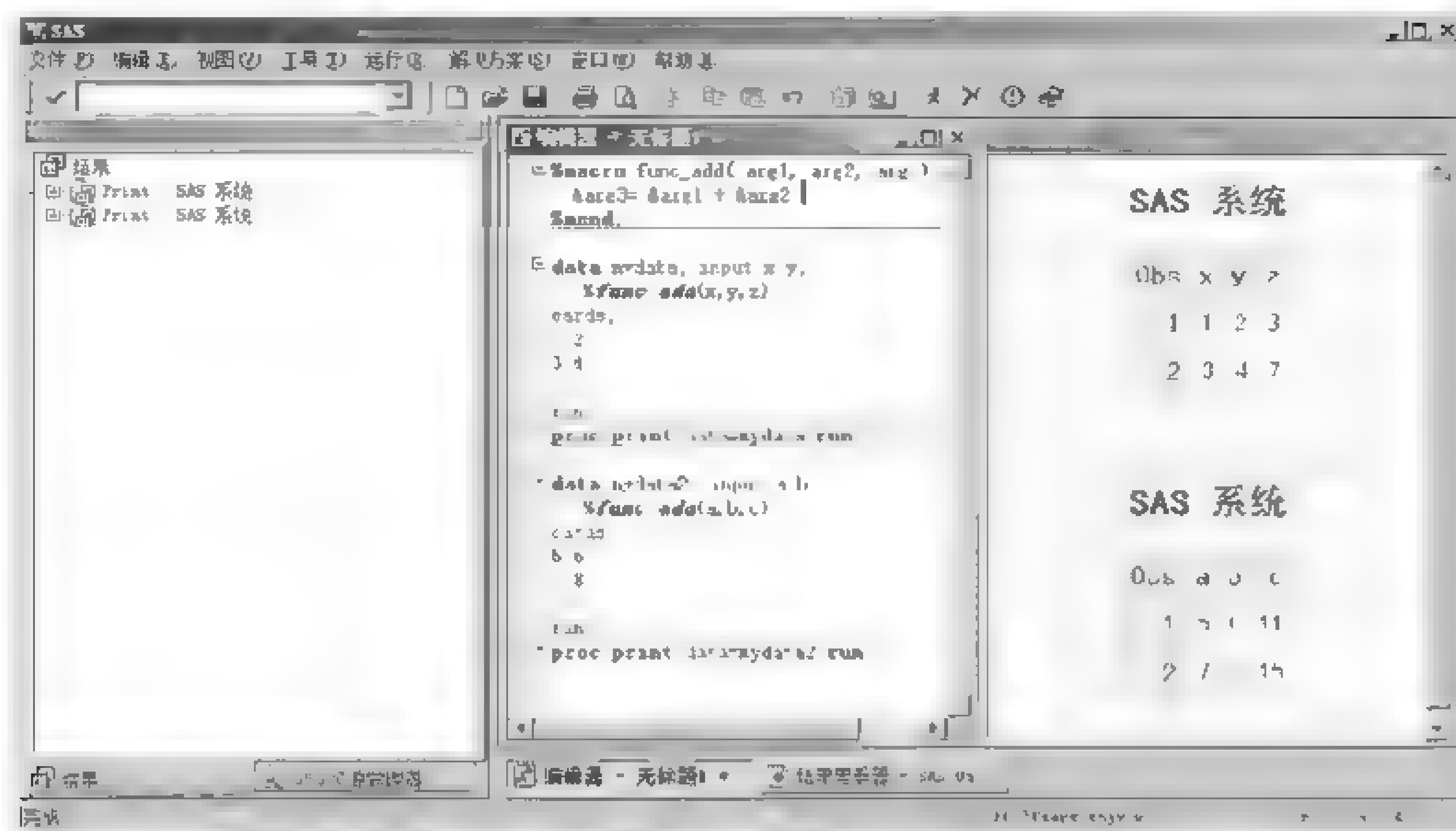


图 5-4　宏函数实现加法运算

除了上面在变量级别进行宏函数封装以外，也可以在数据集的层面上进行 SAS 宏函数封装。此时甚至不必在 SAS 数据步内进行调用，在数据步外面进行控制即可（见程序 5-9）。

程序5-9　将整个 DATA步封装为SAS宏

```
%macro func_add( arg1, arg2, arg3,ds, ds2);
  %if &ds = %then %do ;
    %let ds=&SYSLAST;/*默认使用最近使用的数据集*/
  %end;

  %if &ds2 = %then %do ;
    %let ds2=&ds;/*默认输出结果到原来的数据集*/
  %end;

  data &ds2;
    set &ds;
    &arg3= &arg1 + &arg2;
  run;
%mend;

data mydata;
  input x y;
cards;
1 2
3 4
;
run;

%func_add(x,y,z); /*未指定输出数据集时，直接将结果写入源数据集*/
proc print; run;
```

如果有另一个数据集 mydata2，包含变量 a 和 b，对它的变量进行计算，并将结果输出到 mydata3 中，指定输入数据集参数 ds 和输出数据集参数 ds2 即可（见程序 5-10）。

程序5-10 参数化调用SAS宏函数

```
data mydata2;
  input a b;
cards;
5 6
7 8
;
run;

%func_add(a,b,c,mydata2,mydata3); /*指定输入数据集和输出数据集*/
proc print; run;
```

SAS 宏非常强大，SAS 程序员掌握这些封装技巧对于写出规范的 SAS 代码极其重要。需要强调的一点是，任何形式的 SAS 宏函数封装，展开后的 SAS 代码必须符合 SAS 语言规范，否则可能出现各种错误。很多 SAS 程序员常犯的一个错误是，在一个数据步内调用另一个宏函数，而该宏函数展开实际已经包含一个完整的数据步语句，则 SAS 编译器会因为两个数据步语句嵌套不符合语法而报语法错误。

5.3 FCMP 函数

从 SAS 9.1 版本开始，SAS 提供过程步 PROC FCMP 专门用于函数封装。正如其名字 FCMP（Function Compiler）所揭示的，PROC FCMP 用来专门编译用户自定义函数、CALL 例程以及用户自定义函数库。PROC FCMP 封装的函数可在数据步内自由调用，具有非常好的通用性。虽然某些过程步如 PROC IML 也提供函数封装功能，但其作用范围较为有限。由于 PROC FCMP 封装的函数与 SAS 系统函数在作用范围和调用方式上最为相似，因此它是 SAS 中封装函数的主要手段。

首先，使用 PROC FCMP 过程步定义的函数，编译后输出一个包含指令序列的 SAS 数据集格式的文件。单个 SAS 函数库可以包含多个函数包（Package），而每个函数包则可以包含若干函数（Function）。因此，PROC FCMP 的 outlib 参数采用 3 级名称体系：*libname.dataset.package*。函数编译结果默认是以明文形式存储的，如果需要隐藏存在于 VALUE 列中的明文代码，用户需要启用该过程步的 HIDE 选项来保护自定义函数实现。下面是用 FCMP 封装的加法和乘法函数示例（见程序 5-11）。

程序5-11 FCMP 函数示例

```
/*利用PROC FCMP 来定义函数逻辑，输出到 work.funcs 数据集的 math 包中*/
proc fcmp outlib work.funcs.math;
  function add(arg1, arg2); /*封装加法计算逻辑*/
    ret arg1+arg2;
    return(ret);
  endsub;
  function multiply(arg1, arg2); /*封装乘法计算逻辑*/
```

```
    ret=arg1*arg2;
    return(ret);
  endsub;
run;
```

生成的 FCMP 数据集（如 work.funcs）也可使用 listfuncs 选项来查看函数定义的原型或签名，其信息包括函数名称、类型、返回值类型、参数序列等（见程序 5-12）。

程序5-12　查看 FCMP 函数包的方法/例程的签名信息

```
proc fcmp inlib=work.funcs listfuncs ;
run;
```

系统输出如图 5-5 所示。

```
            FCMP Function/Subroutine Listing
Name      Type           Returns  Prototype
multiply  FCMP Function  Num      multiply( arg1, arg2 ),

add       FCMP Function  Num      add( arg1, arg2 ),
```

图 5-5　FCMP 函数或例程函数原型

也可以直接查看该数据集的详细内容（见图 5-6）。多个 FCMP 过程步可以输出同名的函数到同一数据集的不同函数包。

Obs	_Key_	Owner	Sequence	Type	Subtype	Name	Continue	NValue	Encoded	Value
1	MATH	CMP	0	Header	Package		0	.	.	1830961567.2270
2	F.MATH.MULTIPLY	CMP	0	Prototype	FCmp	math	0	.	.	0012880000819200064320000819200064 32
3	F.MATH.MULTIPLY	CMP	1	Header	Function		0	.	.	1830961567.2270
4	F.MATH.MULTIPLY	CMP	2	Statement Source	Executable	FUNCTION	0	65		function multiply(arg1, arg2)
5	F.MATH.MULTIPLY	CMP	3	Statement Source	Comment	CMT	0	101	.	/*封装乘法计算逻辑*/
6	F.MATH.MULTIPLY	CMP	4	Statement Source	Executable	ASSIGN	0	1		ret=arg1*arg2;
7	F.MATH.MULTIPLY	CMP	5	Statement Source	Executable	RETURN	0	1	.	return(ret);
8	F.MATH.MULTIPLY	CMP	6	Statement Source	Executable	ENDSUB	0	14	.	endsub;
9	F.MATH.ADD	CMP	7	Prototype	FCmp	math	0	.	.	001288000081920006432000081920006432
10	F.MATH.ADD	CMP	8	Header	Function		0	.	.	1830961567.2270
11	F.MATH.ADD	CMP	9	Statement Source	Executable	FUNCTION	0	65	.	function add(arg1, arg2)
12	F.MATH.ADD	CMP	10	Statement Source	Comment	CMT	0	101		/*封装加法计算逻辑*/
13	F.MATH.ADD	CMP	11	Statement Source	Executable	ASSIGN	0	1		ret=arg1+arg2;
14	F.MATH.ADD	CMP	12	Statement Source	Executable	RETURN	0	1	.	return(ret);
15	F.MATH.ADD	CMP	13	Statement Source	Executable	ENDSUB	0	14	.	endsub;

图 5-6　编译后的代码表现为数据集

一旦编译生成 SAS 数据集，就可以在任何需要该计算逻辑的地方引用这个函数。SAS 系统选项 cmplib（即 Compilation Library）用于指定 SAS 程序需要引用的一个或多个由函数库组成的 SAS 数据集。下面的代码（见程序 5-13）对输入数据的前两列 x 和 y 用加法生成 z 列，后两列 a 和 b 用乘法生成 c 列（见图 5-7）。

程序5-13　调用编译好的 FCMP 函数

```
options cmplib=work.funcs;

data mydata;
```

```
  input x y a b;
  z=add(x,y); /*使用我们自己定义的加法函数*/
  c=multiply(a,b); /*使用我们自己定义的乘法函数*/
cards;
1 2 5 6
3 4 7 8
;
run;
proc print data=mydata;run;
```

Obs	x	y	a	b	z	c
1	1	2	5	6	3	30
2	3	4	7	8	7	56

图 5-7 使用编译好的 FCMP 函数

FCMP 函数默认只能有一个返回值，当需要有多个返回值时，可以使用多个 outargs 语句来指定需要返回的参数，从而实现单个函数中返回多个返回值。当然，也可以使用 outargs 语句来指定返回数组参数。程序 5-14 对一个给定的输入值，按照黄金分割比例分割为两个数值 ret1 和 ret2 返回。outargs 语句修饰函数参数机制为从 SAS 函数中返回多个数值成为可能。

程序5-14 从 FCMP 函数返回多个返回值

```
/*按照黄金分割比例分割 arg1为ret1, ret2, 其中两者的比例接近于0.618*/
proc fcmp outlib=work.funcs.math;
  function decomposite( arg1, ret1, ret2);
    outargs ret1;
    outargs ret2;

    theta=(1 + sqrt(5))/2;
    ret1= arg1 / (1 + theta)
    ret2= arg1 / (1 + theta) * theta;
    return (1);
  endsub;
run;

options cmplib=work.funcs; /*调用代码开始*/
data _null_;
  v1=5;
  v2=.; v3=.;
  rc=decomposite(v1, v2, v3);
  r=v3/v2;
  put v1= v2= v3= r= ;
run;
```

输出：

v1=5 v2=3.0901699437 v3=1.9098300563 r=1.6180339887

利用 outargs 语句配合数组参数可从 FCMP 函数中返回批量数据，也就是说 outargs 语句修饰某个参数，实际上表示该参数是“按引用”传递，而非“按值”传递。如果参数为数组时不需要指定数组大小，用 a[*] 或 a[*] $ 表示即可（其中 $ 表示该参数是字符型数组）。程序 5-15 展示了如何在 FCMP 函数中将字符数组颠倒过来。

程序5-15　从FCMP 函数返回一个数组

```
proc fcmp outlib work.funcs.math;
  function reverserarray( args_c[*] $);
    outargs args c;

    length=dim(args c);
    do i=1 to floor(length/2);
      j=length-i+1;
      tmp=args c[i];
      args c[i]= args c[j];
      args c[j]=tmp;
    end;
    return (1);
  endsub;
run;

options cmplib=work.funcs;
data _null_;
  array a[5] $ ("a", "b", "c", "d", "e");

  do i=1 to dim(a);
    put a[i] @@;
  end;
  rc=reverserarray(a);/*调用自定义函数*/
  put "-> " @@;
  do i=1 to dim(a);
    put a[i] @@;
  end;
run;
```

系统输出：a b c d e -> e d c b a

在 FCMP 函数内也可定义本地临时数组，如果该数组是数值型的话，可调用 CALL dynamic_array 例程在程序运行时动态改变该数组长度。然而 SAS 不支持在 PROC FCMP 函数中改变一个从函数外传入数组的大小，也不支持从 PROC FCMP 函数中返回一个本地临时数组给外部函数调用者。

当需要引用多个函数库时，可在 cmplib 中用空格或逗号分隔来指定多个参数，也可以使用减号“-”来表示一系列参数。如果名称相同的函数在多个库中都有定义，则越后指定的库中所定义的函数具有越高优先调用权，这一特性可实现 FCMP 函数重写机制，替换某些现有函数的具体实现。比如：

```
options cmplib=(work.math1 work.math2 work.math3);  /*逐一指定*/
options cmplib=(work.math1 - work.math3);/*批量指定 math1,math2,math3*/
```

当分析人员在分工合作编写大型数据分析项目时，可能在定义函数时需要引用别人写的代码。此时需要在 PROC FCMP 中使用 library 参数来引用别人写好的 SAS 函数库。比如下面的例子创建一个计算平方和的函数 square，它需要引用前面例子中 work.funcs 函数库中已经定义的加法 add 和乘法 multiply 函数（见程序 5-16）。

程序5-16　基于已有的FCMP 函数库开发自定义FCMP 函数

```
proc fcmp outlib work.funcs2.math library=work.funcs ;
  function square(arg1, arg2); /*封装平方和x^2 + y^2 计算逻辑*/
    ret add(multiply(arg1, arg1), multiply(arg2, arg2))  ;
    return(ret);
```

```
  endsub;
run;

/*函数调用阶段：当调用funcs2 中的函数时，需要将它所链接的函数库funcs 也一并包含*/
options cmplib=(work.funcs2 work.funcs);
data mydata;
  input x y;
  z=square(x,y);
cards;
1 2
3 4
;
run;
proc print data=mydata;run;
```

系统输出如图 5-8 所示。

Obs	x	y	z
1	1	2	5
2	3	4	25

图 5-8　链接底层函数库

如果只想定义没有返回值的函数（在 SAS 里称为子例程）该怎么办？只需在定义中将 FUNCTION 改为 SUBROUTINE，不要使用 RETURN 语句指定返回值。此时调用该用户子例程需要在前面加上 CALL，跟调用系统例程方法一样。

另外，在 PROC FCMP 中不但可以调用系统函数、用户自定义 FCMP 函数，也可调用 SAS 宏函数。程序 5-17 展示了 PROC FCMP 如何通过系统函数 RUN_MACRO 调用用户自定义宏函数，该技巧揭示了如何在一个数据步内调用另一个数据步内定义的计算逻辑。

```
程序5-17  在FCMP 函数中调用SAS 宏函数
%macro Add; /*参数: arg1 arg2 ret1*/
  data _null_;
    ret= &arg1 + &arg2;
    call symput('ret1', ret);/*将计算结果放入全局宏变量 ret1 中*/
  run;
%mend;

proc fcmp outlib=work.funcs.math;
  function add(arg1, arg2);
    rc = run_macro('Add', arg1, arg2, ret1);/*调用 Macro 函数 Add */
    if rc = 0 then return(ret1);/*返回计算结果 ret1*/
    else return(.);
  endsub;
run;

options cmplib=(work.funcs);
data mydata;
  input x y;
  z=add(x,y); /*调用 FCMP 函数 add*/
cards;
1 2
3 4
;
run;
proc print data=mydata;run;
```

输出如图 5-9 所示。

Obs	x	y	z
1	1	2	3
2	3	4	7

图 5-9 FCMP 函数实现跨 DATA 步调用机制

实际上，SAS 语言中可以调用其他计算机语言编写的函数库，如 C 语言或 Java 语言编写的函数库，甚至操作系统的 Win32 API 系统动态库。很多编程人员觉得 SAS 很难学，是因为 SAS 语言在设计上实在太灵活，以至于很多编程人员找不到一个固定的代码框架或设计模式，这些系统互操作内容将在后面章节阐述。

5.4 系统函数速查

SAS 系统中提供约 553 个系统函数供用户调用，用户在进行数据分析时大可不必重新发明轮子，而是应该充分了解这些 SAS 系统函数。笔者将所有的系统函数和子例程（带有 * 号前缀者）从简单到复杂归纳为如下 16 个类别，供读者参考，在编程时还需查找帮助文档获得调用细节。

（1）字符和正则表达式：主要包括处理字符数据的 96 个函数或例程，以及专门用于支持 Perl 正则表达式的 11 个函数。

字符处理（96）：ANYALNUM ANYALPHA ANYCNTRL ANYDIGIT ANYFIRST ANYGRAPH ANYLOWER ANYNAME ANYPRINT ANYPUNCT ANYSPACE ANYUPPER ANYXDIGIT BYTE **CATS *CATT *CATX *COMPCOST *MISSING *SCAN* CAT CATQ CATS CATT CATX CHAR CHOOSEC CHOOSEN COALESCEC COLLATE COMPARE COMPBL COMPGED COMPLEV COMPRESS COUNT COUNTC COUNTW DEQUOTE FIND FINDC FINDW FIRST IFC INDEX INDEXC INDEXW LEFT LENGTH LENGTHC LENGTHM LENGTHN LOWCASE MD5 MISSING MVALID NLITERAL NOTALNUM NOTALPHA NOTCNTRL NOTDIGIT NOTFIRST NOTGRAPH NOTLOWER NOTNAME NOTPRINT NOTPUNCT NOTSPACE NOTUPPER NOTXDIGIT NVALID PROPCASE QUOTE RANK REPEAT REVERSE RIGHT SCAN SHA256 SHA256HEX SHA256HMACHEX SOUNDEX SPEDIS STRIP SUBPAD SUBSTR（left of ）SUBSTR（right of ）SUBSTRN TRANSLATE TRANSTRN TRANWRD TRIM TRIMN TYPEOF UPCASE VERIFY

正则表达式函数或子例程（11）：PRXCHANGE PRXMATCH PRXPAREN PRXPARSE PRXPOSN **PRXCHANGE *PRXDEBUG *PRXFREE *PRXNEXT *PRXPOSN *PRXSUBSTR*

（2）数值表达式判断与数据取整处理（12）：

IFN CEIL CEILZ FLOOR FLOORZ FUZZ INT INTZ ROUND ROUNDE ROUNDZ TRUNC

（3）日期时间函数（49）：用来处理各种日期时间类型的数据

IS8601 CONVERT *DATDIF* DATE DATEJUL DATEPART DATETIME DAY DHMS HMS HOLIDAY HOLIDAYCK HOLIDAYCOUNT HOLIDAYNAME HOLIDAYNX HOLIDAYNY HOLIDAYTEST HOUR INTCINDEX INTCK INTCYCLE INTFIT INTFMT INTGET INTINDEX INTNX INTSEAS INTSHIFT INTTEST JULDATE JULDATE7 MDY MINUTE MONTH NWKDOM QTR SECOND TIME TIMEPART TODAY TZONEID TZONENAME TZONEOFF TZONES2U TZONEU2S WEEK WEEKDAY YEAR YRDIF YYQ

（4）数组函数（3）以及数据的查找和排序（4）：

DIM HBOUND LBOUND

WHICHC WHICHN *SORTC *SORTN*

（5）数学函数（39）：ABS AIRY BETA *LOGISTIC *SOFTMAX *STDIZE *TANH* CNONCT COALESCE COMPFUZZ CONSTANT DAIRY DEVIANCE DIGAMMA ERF ERFC EXP FACT FNONCT GAMMA GCD IBESSEL JBESSEL LCM LGAMMA LOG LOG1PX LOG10 LOG2 LOGBETA LOGISTIC MOD MODZ MSPLINT SIGN SQRT TNONCT TRIGAMMA 以及一个特殊的除法函数 DIVIDE

三角函数 （10）：ARCOS ARSIN ATAN ATAN2 COS COT CSC SEC SIN TAN

双曲函数（6）：ARCOSH ARSINH ARTANH COSH SINH TANH

（6）SAS 宏有关操作（9）：

*EXECUTE *SYMPUT *SYMPUTX* DOSUBL RESOLVE SYMEXIST SYMGET SYMGLOBL SYMLOCAL

（7）字节按位操作函数（6）：BAND BLSHIFT BNOT BOR BRSHIFT BXOR

（8）文件和目录通用操作（37）：DCLOSE DCREATE DINFO DNUM DOPEN DOPTNAME DOPTNUM DREAD DROPNOTE DSNCATLGD FAPPEND FCLOSE FCOL FDELETE FEXIST FGET FILEEXIST FILENAME FILEREF FINFO FNOTE FOPEN FOPTNAME FOPTNUM FPOINT FPOS FPUT FREAD FREWIND FRLEN FSEP FWRITE MOPEN PATHNAME RENAME SYSMSG SYSRC

SAS 数据文件专用操作（33）：ATTRC ATTRN CEXIST CLOSE CUROBS DROPNOTE DSNAME ENVLEN EXIST FCOPY FETCH FETCHOBS GETVARC GETVARN IORCMSG LIBNAME LIBREF NOTE OPEN PATHNAME POINT RENAME REWIND SYSEXIST SYSMSG SYSRC VARFMT VARINFMT VARLABEL VARLEN VARNAME VARNUM VARTYPE

（9）变量信息（31）和变量控制（3）：

**VNEXT* VARRAY VARRAYX VFORMAT VFORMATD VFORMATDX VFORMATN VFORMATNX VFORMATW VFORMATWX VFORMATX VINARRAY VINARRAYX VINFORMAT VINFORMATD VINFORMATDX VINFORMATN VINFORMATNX VINFORMATW VINFORMATWX VINFORMATX VLABEL VLABELX VLENGTH VLENGTHX VNAME VNAMEX VTYPE VTYPEX VVALUE VVALUEX

*LABEL *SET *VNAME

（10）内存读写及操作系统相关的特殊函数（31）：

ADDR ADDRLONG *POKE *POKELONG *SLEEP *SYSTEM *TSO DIF FMTINFO GETOPTION INPUT INPUTC INPUTN LAG PEEK PEEKC PEEKCLONG PEEKLONG PTRLONGADD PUT PUTC PUTN SLEEP SYSEXIST SYSGET SYSPARM SYSPROCESSID SYSPROCESSNAME SYSPROD SYSTEM UUIDGEN

调用外部模块（4）：*MODULE MODULE MODULEC MODULEN

安装模块检查（1）：MODEXIST

（11）HTML 与 URL 编解码（4）和 SOAP Web 服务（6）：

HTMLDECODE HTMLENCODE URLDECODE URLENCODE SOAPWEB SOAPWEBMETA SOAPWIPSERVICE SOAPWIPSRS SOAPWS SOAPWSMETA

（12）随机数处理（22）：*RANBIN *RANCAU *RANEXP *RANGAM *RANNOR *RANPOI *RANTBL *RANTRI *RANUNI *STREAMINIT NORMAL RANBIN RANCAU RAND RANEXP RANGAM RANNOR RANPOI RANTBL RANTRI RANUNI UNIFORM

（13）排列组合（23）：ALLCOMB ALLPERM *ALLCOMB *ALLCOMBI *ALLPERM *GRAYCODE *LEXCOMB *LEXCOMBI *LEXPERK *LEXPERM *RANCOMB *RANPERK *RANPERM COMB GRAYCODE LCOMB LEXCOMB LEXCOMBI LEXPERK LEXPERM LFACT LPERM PERM

（14）描述性统计（32）：CMISS CSS CV EUCLID GEOMEAN GEOMEANZ HARMEAN HARMEANZ IQR KURTOSIS LARGEST LPNORM MAD MAX MEAN MEDIAN MIN MISSING N NMISS ORDINAL PCTL RANGE RMS SKEWNESS SMALLEST STD STDERR SUM SUMABS USS VAR

概率（18）与分位数（8）：CDF LOGCDF LOGPDF LOGSDF PDF POISSON PROBBETA PROBBNML PROBBNRM PROBCHI PROBF PROBGAM PROBHYPR PROBMC PROBNEGB PROBNORM PROBT SDF BETAINV CINV FINV GAMINV PROBIT QUANTILE SQUANTILE TINV

（15）邮政编码（12）和地理空间（2）有关函数：FIPNAME FIPNAMEL FIPSTATE STFIPS STNAME STNAMEL ZIPCITY ZIPCITYDISTANCE ZIPFIPS ZIPNAME ZIPNAMEL ZIPSTATE GEODIST ZIPCITYDISTANCE

（16）金融计算（41）函数：

BLACKCLPRC BLACKPTPRC BLKSHCLPRC BLKSHPTPRC COMPOUND CONVX CONVXP CUMIPMT CUMPRINC DACCDB DACCDBSL DACCSL DACCSYD DACCTAB DEPDB DEPDBSL DEPSL DEPSYD DEPTAB DUR DURP EFFRATE FINANCE GARKHCLPRC GARKHPTPRC INTRR IPMT IRR MARGRCLPRC MARGRPTPRC MORT NETPV NOMRATE NPV PMT PPMT PVP SAVING SAVINGS TIMEVALUE YIELDP

第6章

SAS宏

在传统编程语言C/C++中，可以使用预处理语句#define来定义宏变量，也可以使用其他以#号开头的宏语句来实现条件编译功能，如#ifdef….#else… #endif等。也有一些语言如Java不提供预处理支持，编程人员只好使用静态常量以及泛型技术来实现类似控制机制。C#语言为了灵活的编译控制依然保留了#define、#if、#else、#endif等宏语句。在计算机语言中，宏到底是什么，它在SAS语言中又是如何工作的呢？

总体来说，宏技术在编译领域的作用就是告诉编译器在正式代码编译之前，由预处理器根据一系列预定义的规则有条件地对源代码进行文本替换，称之为宏展开。宏技术是编程语言之上的更大范围的一种高级抽象（据《康熙字典》：宏賫皆大也），它能在比源代码更高的层次上提供灵活的逻辑控制能力，即源代码进入编译阶段之前执行一系列预处理功能。

SAS宏比传统编程语言的宏技术“有过之而无不及”，实现了SAS代码的高级复用，让SAS程序更加精巧和灵活。SAS宏易于学习，但也很容易令人困惑，因为程序员很容易将宏本身，跟宏展开后输出代码混为一谈。SAS宏系统非常强大，不但支持宏变量替换和条件分支控制，而且支持更加高级的循环控制，以及宏函数（系统宏和用户自定义宏）机制。宏技术本身在SAS语言里已经发展成了一种特殊语言，可在更抽象的层面上对SAS源代码进行操纵，实现分析数据、编写报告以及自动执行代码等功能。

通过前面章节的学习我们知道SAS程序由各种SAS语句，包括全局语句、DATA步或PROC步语句组成；但SAS代码中其实还可以包含SAS宏语言、SCL语言和SQL语言编写的语句。SAS代码提交执行时，源代码首先被读入计算机内存的输入缓冲区中进行字符扫描，如果字符包含SAS宏技术约定的字符&和%（前者标识宏变量，后者标识宏语句）时，这些代码就会触发宏处理器进行宏展开操作，宏展开后的代码会追加到输入缓冲区供编译器继续进行处理。

宏展开完毕的SAS代码再交由SAS编译器进行词法分析和编译运行。需要特别注意一点是，SAS宏技术的引入并未减少SAS程序的执行时间，而是减少编写重复或高相似性的SAS代码，从而增强了SAS程序的可维护性。SAS代码预处理机制如图6-1所示，宏处理器负责解析SAS宏代码。



图6-1　SAS代码预处理机制

6.1 宏变量

宏变量本质上就是一种字符类型的变量，由它的名字和字符串值共同构成一个符号表。宏变量的值可以来自真正的SAS变量，数字或文本，或者是宏表达式。SAS源代码中出现%号和&号的地方（包括代码中以双引号括起来的字符串中）都是触发SAS宏处理的地方，但单引号括起来的字符串中出现的%号和&号不会触发宏展开。

6.1.1 命名

SAS宏变量命名遵循SAS命名规则，可以是字母、数字或下划线，但必须以字母或下划线开头，最大长度不得超过32字节。如果定义宏变量的时候该宏变量在系统符号表中已经存在，系统会自动覆盖已有的宏变量。

宏变量可以使用%GLOBAL或%LOCAL语句预先定义，也可以直接用%LET语句定义并赋值。其中%LET语句主要用于赋值，仅在宏变量没有定义时才会自动创建。当前SAS会话的任何代码（包括DATA步外的开放代码）中，都可使用&宏变量名方式对宏变量进行引用。其基本语法如下：

```
%LET MACRO-VARIABLE =<VALUE>;
```

比如：

```
%let foo=value;
%put &foo;
```

系统输出：VALUE

当使用%LET语句对宏变量进行赋值时，其等号右侧的内容被SAS处理时遵循如下规则。

（1）首尾的连续空格在赋值前会被自动删除。

（2）数值也是被当作文本看待，数学表达式不会被自动求值，除非使用特定宏函数处理。

（3）引号本身也是宏的一部分，宏表达式中的引号并不是字符串标记符号，只是作为字符串的引号本身。

（4）文本长度必须介于1~65534字符，比16位无符号整数的最大值65535少1。

6.1.2 作用域

宏变量具有作用域的概念，宏变量可以在SAS会话中全局有效，或者在SAS宏函数中局部有效。如果宏变量定义在一个SAS宏函数内部，它的作用域就是局部的，只能在该宏函数内起作用。但如果宏变量定义在任何宏函数外，则可在SAS程序任何地方使

用已经定义的全局宏变量。考察如下代码：

```
%put &foo; /*试图引用一个不存在的宏变量*/
```

输出：

```
WARNING: Apparent symbolic reference FOO not resolved.
&FOO
```

如果改为

```
%let foo=GLOBAL "MACRO" VAR;
%put &foo; /*输出: GLOBAL "MACRO" VAR*/
```

如果%LET语句在一个SAS宏函数内部给某个宏变量赋值，默认情况下该宏变量是宏函数的局部变量。如果%LET语句在非宏函数内的开型代码中，则该宏变量为全局变量。

在调用宏变量赋值语句%LET之前，可使用%GLOBAL和%LOCAL语句先行对宏变量进行定义，分别指定宏变量作用域是全局还是局部。如果已经定义了某个全局宏变量，同时在某个宏函数内部再次用%LOCAL定义一个同名宏变量，则宏处理器在执行到该宏函数内时使用局部定义的宏变量值，在该宏函数之外使用全局宏变量值。这说明局部宏本质上是**定义在宏函数局部的符号表中，而非全局符号表中**；宏变量的查找顺序也是先从宏函数局部符号表查找，然后再从全局符号表中查找。下面的代码（见程序6-1）将输出：GLOBAL GLOBAL LOCAL GLOBAL，最后一个输出的是GLOBAL而不是宏函数MyMacroFunction内设置的值LOCAL。该代码可帮助读者很好地理解SAS宏变量的作用域。

程序6-1　理解 SAS宏变量的作用域

```
%let foo=GLOBAL;
%put &foo; /*输出: GLOBAL*/
%macro MyMacroFunction;
  %put &foo; /*输出: GLOBAL*/

  %local foo;
  %let foo=LOCAL;
  %put &foo; /*输出: LOCAL*/
%mend;
%MyMacroFunction;
%put &foo; /*输出: GLOBAL*/
```

6.1.3　系统宏

系统宏是预先定义在全局符号表中的一系列宏变量，它们在当前SAS会话中全局有效。比如全局宏变量&SYSDATE和&SYSTIME，分别表示当前SAS会话开始的日期和时间。检测当前SAS会话中到底有哪些宏被定义了，可调用宏语句%PUT来显示。比如：

```
%put _ALL_;        /*列出所有宏变量*/
%put _AUTOMATIC_;  /*列出所有系统定义的宏变量*/
%put _GLOBAL_;     /*列出所有全局(会话级)宏变量*/
%put _LOCAL_;      /*列出所有局部(正在执行的宏)宏变量*/
%put _USER_;       /*列出所有用户定义的宏变量*/
```

如果对已经定义的宏变量使用 & 宏变量名进行引用，但 SAS 系统约定对包含在单引号内的文本不会进行宏展开。程序员可在代码中直接引用宏变量，也可在双引号文本中引用宏变量。考察如下代码（见程序 6-2）：

程序 6-2　宏变量解析在单引号文本中被忽略

```
%let dsname=sashelp.class;
%let author=Yinliang Wu;
title "Content of dataset &dsname "; /*双引号中引用宏变量*/
title2 'Author:  &author'; /*单引号中不作宏展开*/
proc print data=&dsname; /*直接引用宏变量*/
run;
```

上面的代码会输出期望的报表标题，当 title2 语句指定次一级标题时，宏变量 &author 并不起作用（见图 6-2）。原因是在 title2 语句中使用了单引号来包含字符串，而单引号字符串中的 & 和 % 号是不会被宏展开的，此时换成双引号即可。

Content of dataset sashelp.class
Author: &author

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
2	Alice	F	13	56.5	84.0
3	Barbara	F	13	65.3	98.0

图 6-2　宏变量仅在双引号中起作用

通常宏变量引用以 & 符开始，以空格结束；有时为了在连续的文本中明确指示宏变量的起止，可使用 & 符开始，以英文句号 . 符结束的形式来明确区分宏变量与前后文本，如程序 6-3 所示。

程序 6-3　宏变量的起止控制

```
%let var1=leon;
data _null_;
  put "&var1a and &var1ardo";
run;
```

运行代码后系统报告没有解析符号引用 VAR1A 和 VAR1ARDO，而实际上我们是希望输出 *Leona and Leonardo*，此时可以将 put 语句修改如下即可：

```
put "&var1.a and &var1.ardo";
```

& 符号作为宏变量的标识，如果需要生成 & 字符本身，则需要使用两个连续的 & 符代替。基于这一特性，可以定义多重宏变量引用，实现多层宏变量的逐步展开。比如程序 6-4 的 %PUT 语句输出都是 Leon。

程序 6-4　多层宏变量解析和引用

```
%let var1=leon;
%let var2=&&var1;
%put &var1 &var2;
```

6.1.4 宏代码调试

SAS 主要提供两个系统选项 mprint 和 mlogic 来帮助我们调试宏代码，用于静态检查宏展开生成的代码及动态跟踪宏的执行过程。

（1）静态检查宏展开代码：请运行如下代码（见程序 6-5）并检查日志窗口。

程序6-5 启用 SAS 宏的静态调试选项

```
options mprint; /*是否打印宏展开生成的 SAS 语句代码? 缺省为 NOMPRINT*/
options mprintnest; /* 打印宏展开代码时是否嵌套显示? 缺省为 NOMPRINTNEST*/

%macro printclass();
  proc print data=sashelp.class;
  run;
%mend;
%printclass;
```

如果我们希望把 SAS 宏展开的代码输出到特定的文件中调试，可以在宏 %macro printclass 之前插入如下代码（见程序 6-6），则由宏展开后的源代码会被输出到外部文件 C:\mymacro.sas 中。

程序6-6 将 SAS 宏展开结果输出到源代码文件

```
options mfile; /*打印时是否输出到外部文件, 缺省为 NOMFILE*/
filename mprint 'C:\mymacro.sas';
/*若已启用 mprint 和 mfile 系统选项，且用filename语句指定了文件引用 mprint，则代码会自动输出到外部文件 */
```

（2）动态跟踪宏执行过程：可启用系统选项 mlogic 来跟踪宏处理器的动态执行逻辑（见程序 6-7）。

程序6-7 启用跟踪宏处理器的动态执行逻辑

```
options mlogic; /*是否跟踪宏处理器执行过程? 缺省为 NOMLOGIC*/
options mlogicnest; /*mlogic 是否显示嵌套? 缺省为 NOMLOGICNEST*/
```

6.1.5 宏表达式

与 DATA 步中的 SAS 表达式类似，SAS 宏也支持表达式，称为宏表达式。它可用于求值运算或逻辑控制。SAS 宏表达式包含算术运算符、比较运算符和逻辑运算符。SAS 宏表达式在语义上跟 SAS 表达式基本一致，有相当多的 SAS 语句加上 % 号即可从普通 SAS 代码变为 SAS 宏代码。使用 SAS 宏表达式需要注意如下一些事项：

（1）宏代码中的比较和逻辑运算符不必且不能使用百分号，包括 LT、GT 和 AND、OR 等。

（2）宏表达式不支持形如 *50.0 <= &Weight <= 80.0* 的判断条件，必须用多个比较表达式并用 AND 连接起来。

（3）宏表达式中可以有括号 ( ) 进行分组，但有时省略括号 ( ) 也可正常工作（见程序 6-8）。

程序6-8　宏表达式范例

```
%macro test;
  %let a=8;
  %if 0<&a and (&a <5) %then %put inside;
  %else %put outside;
%mend;
%test;
```

由于宏展开的本质是文本替换，宏里面的字符内容不需要使用单引号或双引号括起来，且宏表达式作为字符串的值时是大小写敏感的（见程序 6-9）。

程序6-9　给宏变量赋值时大小写敏感

```
%let a=Hello World;
%let b=HELLO WORLD;
%let c="Hello World";
%put &a &b &c; /* 输出：Hello World HELLO WORLD "Hello World"*/
```

宏变量只能是文本，即使是数值也只能以文本方式存在。为了在 SAS 宏表达式中支持算术运算、比较运算和逻辑运算，SAS 提供两个系统宏函数 %EVAL 和 %SYSEVALF 对宏表达式进行求值。两者的区别是前者只支持不包含小数点的整数表达式；而后者 %SYSEVALF 可支持浮点运算，即宏表达式或求值结果中可以包含小数点，也可以指定转换类型。如果表达式中不包括运算符，则宏函数会直接返回表达式原来的值。两个系统宏函数的语法如下：

```
%EVAL (EXPRESSION)
%SYSEVALF (EXPRESSION, CONVERSION-TYPE )
```

考察如下代码（见程序 6-10）：

程序6-10　对宏表达式进行求值

```
%let exp=4/3;
%let eval_V1=%eval(&exp);
%put &eval_V1; /*输出：1*/

%let exp1=3;
%let exp2=4;
%put %sysevalf( &exp1 * &exp1  + &exp2 * &exp2 ); /*输出平方和：25*/

%let eval_V2=%sysevalf(&exp); /*输出：1.3333333333333而非 1*/

%let sysevalf_V1= %sysevalf(2>1,boolean);      /* 输出：1 */

%let sysevalf_v4= %sysevalf(5.49,ceil);        /* 输出向上取整：6 */
%let sysevalf_v5= %sysevalf(5.49,integer);     /* 输出四舍五入：5 */
%let sysevalf_v6= %sysevalf(5.49,floor);       /* 输出向下取整：5 */

%let sysevalf_v4x= %sysevalf(-5.49,ceil);      /* 输出向上取整：-5 */
%let sysevalf_v5x= %sysevalf(-5.49,integer);   /* 输出四舍五入：-5 */
```

在编写宏代码运行时可能会遇到错误：“ERROR：　检测到开型代码语句的递归”，这往往是因为宏语句忘了用分号结束，导致宏展开无法继续。

6.2 宏函数

宏变量比较简单也容易理解，但SAS宏函数就相对复杂。比如需要对一系列的数据集执行一系列相同或类似的SAS分析代码时，可以考虑利用宏函数来封装“重复或类似的SAS代码”。SAS程序员不必使用复制/粘贴使源代码非常冗长，像封装函数一样对SAS代码进行高层次的代码封装即可，宏展开发生在SAS代码编译之前。

SAS宏函数定义的基本语法如下：

```
%MACRO MACRO-NAME <(PARAMETER-LIST)> </ OPTION-1 <… OPTION-N>> ;
   MACRO STATEMENTS;
%MEND;
```

在宏定义%MACRO语句和%MEND语句之间，可以包括任何SAS语句（如DATA/PROC步），也可以包含宏变量引用，宏语句或表达式以及对其他宏的调用，甚至可以包括纯文本，代码注释等。唯一准则就是宏展开后的文本必须符合SAS语言的语法规范。

常见的一个错误是，企图在一个DATA步内调用一个宏，而该宏已经封装了一个或多个DATA步的调用，这时系统就会编译不过。原因很简单，宏展开后会导致DATA步代码嵌套，从而导致语法错误。另外，SAS宏封装的代码中不建议使用行注释，而应该尽可能使用以“/*”开始，以“*/”结束的块注释，以防止不期望的宏展开。

对于一个已经定义好的宏函数，可以在程序中除了DATA步内数据行以外的任何地方进行调用，包括使用%macro-name方式进行调用，在SAS语句中调用和在SAS工作环境的命令行输入窗口中进行调用三种方式。由于SAS宏并非是SAS语句，调用时宏名称后的分号是不必要的，因为分号是SAS语句的结束符，在某些情况下可能会导致宏展开后插入不必要的分号，从而导致最终代码发生编译或运行错误；很多时候在宏调用时加上分号是为了告诉宏解析器宏调用结束，可开始宏展开及随后的编译执行。

```
%macro-name;
```

比如，可以将打印数据集的SAS代码封装成宏，供重复调用（见程序6-11）。

程序6-11 宏函数示例

```
%macro printds;
  title "content of dataset &dsname";
  proc print data=&dsname;
  run;
%mend;

/*调用1：打印 sashelp.class*/
%let dsname=sashelp.class;
%printds;

/*调用2：打印 sashlep.prdsale */
%let dsname sashelp.prdsale;
%printds;
```

运行上面的代码可以看到SAS宏被执行两次，分别输出SASHELP.CLASS和SASHELP.PRDSALE两个数据集的内容。但由于上面的代码需要依赖于全局宏变量&DSNAME，因此代码的耦合性不好。这时就需要使用宏函数的参数列表，对参数进行封装。

参数定义

SAS宏函数的参数可以用两种方式进行定义：一种称为顺序参数或位置参数；另一种称为命名参数或键值参数。

（1）顺序参数：就是定义宏的时候，按照先后顺序定义所需的形式参数；在宏调用时，也需要按照同样的顺序提供实际参数。比如：

```
%MACRO FOO(ARG1, ARG2,… , ARGN);
  MACRO STATEMENTS;
%MEND;
%FOO (V1, V2,…, VN);
```

据此修改前面打印数据集的SAS宏如下，可同时指定报表标题（见程序6-12）：

程序6-12 带有位置参数的宏函数示例

```
%macro printds(dsname, title);
  title "content of dataset &title";
  proc print data=&dsname;
  run;
%mend;
%printds(sashelp.class, Student);  /*调用形式*/
%printds(sashelp.prdsale, Product Sales);
```

（2）命名参数：在宏定义时可给参数命名，然后在实际调用的时候也采用“参数名=参数值”的方法来指定实际参数。由于每个参数已经定义了名称入口，因此函数调用时参数位置就不再重要。在宏函数调用时如果没有指定参数，SAS则会使用宏函数定义时所指定默认值进行调用。其语法如下：

```
%MACRO FOO(ARG1= DEF_V1, ARG2=DEF_V2,…, ARGN= DEF_VN);
  MACRO STATEMENTS;
%MEND;
%FOO (ARG2=V2, ARGN=VN);
```

再次修改前面打印数据集的样例代码，给宏函数PRINTDS指定参数名称和默认值（见程序6-13）。

程序6-13 带有命名参数的宏函数示例

```
%macro printds(dsname=sashelp.class, title=class);
  title "content of dataset &title";
  proc print data=&dsname;
  run;
%mend;

%printds(); /*以宏函数定义的默认值进行调用*/
%printds(title=My Class); /*仅指定 title 参数*/
%printds(dsname=sashelp.prdsale, title=My Product Sales);
```

6.3 逻辑控制

与传统的C/C++宏编译技术不同，SAS不但提供条件分支控制，还提供循环控制。这样SAS宏技术就演变为SAS代码之上的“超级代码”。因此，合理巧妙地利用SAS宏就可以写出简洁优美的SAS程序，但滥用SAS宏则会导致代码可读性差且调试困难。

6.3.1 宏语句块

与SAS代码一样，可以使用%DO-%END将多条SAS宏语句进行分组，功能与DATA步的DO-END类似。

```
%DO;
  MACRO STATEMENT;
%END;
```

6.3.2 条件分支

基于一个或多个条件，选择性地执行条件块里面的代码；与DATA步中的条件分支类似，%ELSE语句是可选的。其基本语法如下：

```
%IF <MACRO EXPRESSION> %THEN <TRUE MACRO STATEMENT>;
%ELSE <FALSE MACRO STATEMENT>;
```

宏语句也可以用上面的宏语句块%DO-%END将多个宏语句包装为单个宏语句，实现嵌套。宏代码中也可使用跟前面SAS代码相同的%ELSE %IF结构实现多分支控制，此处不再赘述。

```
%IF <MACRO EXPRESSION> %THEN %DO;
    TRUE MACRO STATEMENT;
%END;
%ELSE %DO;
    FALSE MACRO STATEMENT;
%END;
```

6.3.3 循环控制

SAS宏的循环控制主要有如下几种形式：

（1）指定次数的循环DO-TO-BY：具有固定循环次数或步长时使用，类似于传统编程中的FOR循环，其中控制循环起点START、终点END和步长STEP可以是宏常量、宏变量或者任何能够展开为整数的宏表达式，而%BY语句是可选，默认步长为1。

```
%DO <MACRO VAR> = <START> %TO <END> [%BY STEP];
  MACRO STATEMENT;
%END;
```

（2）指定条件的循环 DO-WHILE：在进入循环体前进行判断，为真则执行循环体，为假则离开循环。

```
%DO %WHILE <MACRO EXPRESSION>;
  MACRO STATEMENT;
%END;
```

（3）指定条件的循环 DO-UNTIL：执行循环体后进行判断，为真则退出循环体，为假则继续循环。这种循环方式至少会执行循环体一次。

```
%DO %UNTIL <MACRO EXPRESSION>;
  MACRO STATEMENT;
%END;
```

为综合演示 SAS 宏的逻辑控制，下面我们完全用 SAS 宏语言来实现黄金分割数列的计算（见程序 6-14）。

程序6-14　宏版的斐波那契数列生成器

```
options nomprint nomlogic;
%macro Fbnc;
  %local x1 x2;
  %do n=1 %to 10;
    %if %eval(&n=1 or &n=2) %then
      %let x=1;
    %else
      %let x= %eval( &x1 + &x2 );
    %let x2=&x1;
    %let x1=&x;
    %put n=&n x=&x; /*打印到SAS日志*/
  %end;
%mend;
%Fbnc;  /*调用宏函数 Fbnc*/
```

系统输出为

```
n=1 x=1
n=2 x=1
n=3 x=2
n=4 x=3
n=5 x=5
n=6 x=8
n=7 x=13
n=8 x=21
n=9 x=34
n=10 x=55
```

6.4　系统宏函数

SAS 系统提供若干宏函数，可用于处理跟 SAS 宏有关的字符处理和计算，宏展开以及系统函数等。系统宏函数可使用在开型代码和宏函数内，功能主要包括字符处理，宏表达式求值、引号处理、宏变量以及调用 SAS 系统函数等几大类。简要描述如下。

1. 字符宏函数（8）：用于定位子字符、检测长度、扫描字符、取子串以及字符串

大小写转换等。其中首字母以Q开头的宏函数用于结果中需要屏蔽特殊字符和助记符时使用。

```
%INDEX %LENGTH %SCAN %QSCAN %SUBSTR %QSUBSTR %UPCASE %QUPCASE
```

2. 求值宏函数（2）：分别用于整型和浮点型表达式求值。

```
%EVAL %SYSEVALF
```

3. 引号宏函数（8）：用来屏蔽特殊字符和助记符，使宏处理器将它们解释为纯文本，而不是宏语言的元素。其中%STR和%NRSTR用于宏编译时的常量文本，其他大部分用于宏执行阶段的解析值。

```
%BQUOTE %NRBQUOTE %QUOTE %NRQUOTE %STR %NRSTR %SUPERQ %UNQUOTE
```

4. DBCS特定的宏函数(11)：主要由一系列首字母为K源自日文汉字(Kanji的缩写)的宏函数组成，主要功能包括压缩空格、定位字符、取长度、扫描字符、取子串和大小写转换等。

```
%KCMPRES %KINDEX %KLEFT %QKLEFT %KLENGTH %KSCAN %QKSCAN %KSUBSTR
%QKSUBSTR %KUPCASE %QKUPCASE
```

5. 杂项函数（7）：主要功能包括检测宏变量是否存在，宏变量作用域检测，在SAS宏中执行SAS系统函数或用户函数，以及读取操作系统环境变量，SAS产品安装模块检测等，其中调用系统函数，%SYSFUNC非常有用。

```
%SYMEXIST %SYMGLOBL %SYMLOCAL %SYSFUNC %QSYSFUNC %SYSGET %SYSPROD
```

SAS宏为SAS编程语言提供了语言之上的超级语言，它可以让SAS程序在编译前动态改变或生成SAS源代码。SAS宏展开后生成的SAS代码依然要求遵守SAS语言规范，编译器才能够正常编译执行。

第 [illegible] 章

DS2

SAS 传统的 DATA 步虽然强大，但从计算机语言的角度分析其编程特性，仍有诸多不足之处，主要表现在如下几个方面。

（1）基础数据类型有限，仅支持定长字符类型和双精度浮点型两种基本数据类型。虽然在数据分析中不是什么问题，但当 DATA 步与关系数据库管理系统（RDBMS）进行交互时，数据库所支持的多种数据类型可能在与 DATA 步交互时损失精度。

（2）变量没有作用域概念，DATA 步中没有显式的模块化和面向对象编程的基本概念。

（3）运行态默认只有进程级的多任务处理，没有内置的多线程支持机制。缺乏对分布式并行架构的支持，不能充分利用各种 SMP/MPP 硬件架构下对海量数据并行处理和高性能计算能力。

基于以上原因，SAS 从 9.3 版开始扩展传统的 DATA 步语言，设计了第 2 代 DATA 步语言（简称 DS2），并在 9.4 版本开始成为正式的产品特性。DS2 提供了类似面向对象编程的基本概念：它将程序逻辑执行单元封装为方法，将方法和数据封装为对象实体。从而实现 DS2 在继承第一代 DS 语言流程控制和语言表达式风格基础上，向对象方向迈出重要一步。与传统 DATA 步相比，DS2 有一些增强特性如下所述。

（1）提供多达 17 种基本数据类型，支持强大的高精度运算能力。因此它要求变量、方法和对象等程序实体必须显式定义，而不像在 DS 中随用随定义。

（2）程序变量有作用域机制，并提供类似于 Java/C++ 语言的类 / 方法封装机制。

（3）提供线程并行执行特性，能充分利用 SMP/MPP 硬件架构的计算能力。

在具体表现形式上，DS2 程序表现为封装在 PROC DS2 或 PROC HPDS2 过程步中的 SAS 语句集合。程序 7-1 是 DS2 版的 HelloWorld 程序：

程序7-1 DS2 版的HelloWorld 程序

```
proc ds2;
  data _null_;
    method init();
      declare   varchar(32) str;
      str = 'The Power to Know(R)';
      put str;
      put '---- Since 1976 ----';
    end;
  enddata;
run;
quit;
```

在 DS2 中变量必须在程序头部先进行声明，且字符串必须用单引号括起来。DS2 中

不能用双引号表示字符串，因为在DS2中双引号表示变量引用而非通常的字符串。在Base SAS环境中执行上面的代码，系统输出如图7-1所示。

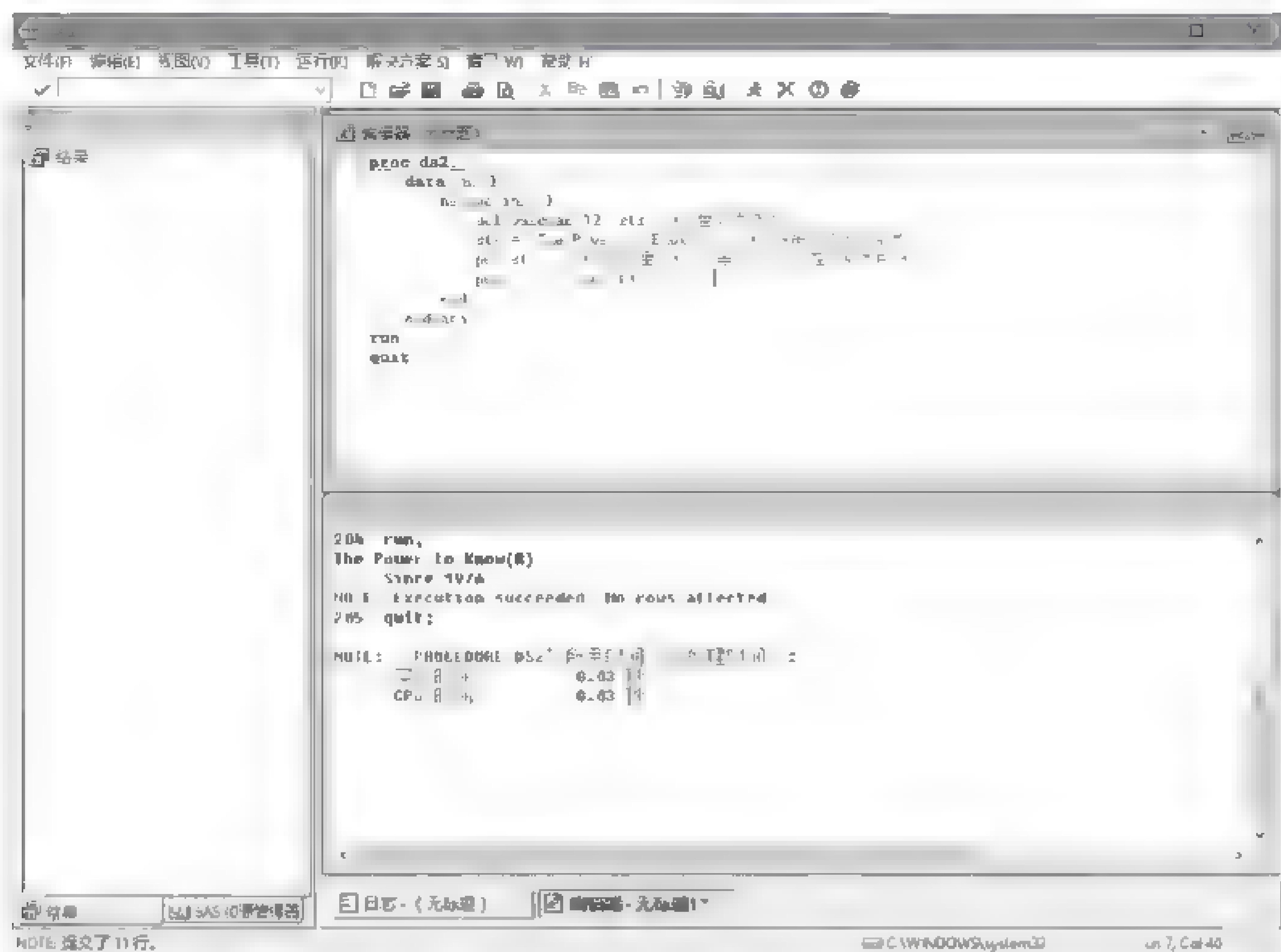


图7-1 DS2版的HelloWorld程序

实际上，DS2程序可以运行在如下4种不同的SAS计算环境中。

（1）BASE环境：即日常运行SAS的基础环境。

（2）HPA环境：即SAS高性能分析环境（SAS High-Performance Analytics）。

（3）INDB环境：即SAS库内分析环境（SAS In-Database Code Accelerator），需要SAS库内代码加速器和SAS嵌入式进程（SAS Embedded-Process）模块的支持。

（4）IMA环境：即SAS内存分析环境（SAS In-MemoryAnalytics），包括SAS最新的CAS环境。

在第一代DS语言中，有LINK-RETURN伪函数、SAS宏函数和FCMP函数3种函数封装技术，其中，FCMP函数封装技术跟C/C++和Java语言中的函数概念最为接近，函数复用性也比较广；模块化代码复用则主要依赖静态文件包含%INCLUDE来实现。

DS2在兼容传统DS语言的基础上，在模块化，健壮性和代码重用方面有飞跃性的进步。其中包与方法封装技术跟作用域概念的引入将SAS这种专门的数据分析语言推到准面向对象时代。面向对象的三大特性为封装、继承和多态性，DS2提供对象封装和方法重载，但不支持继承机制。因此DS2只提供了准面向对象编程概念。

DS2技术的诞生并不是要替换掉传统DS步，而是传统DS步的增强和扩展；DS2增强由于是面向分布式计算架构而设计，因此DS2不再支持某些专为单机系统设计的产品特性，如读取PC数据文件的INPUT语句以及合并数据的MERGE语句等。但DS2和DS之间依然可用SAS数据集和SAS宏紧密联系在一起形成互补的强大编程系统。

7.1 程序结构

DS2 程序表现为封装于 PROC DS2 过程步中的程序块，主要包括 3 种程序类型：

（1）数据程序（DATA PROGRAM），封装于 DATA ... ENDDATA 语句之间。

（2）包程序（PACKAGE PROGRAM），封装于 PACKAGE ... ENDPACKAGE 语句之间。

（3）线程程序（THREAD PROGRAM），封装于 THREAD ... ENDTHREAD 语句之间。

下面的 DS2 数据程序生成只包含一条观测的数据集 D，其中变量 c 的内容为字符串“ds2 data program”（见程序 7-2）。

程序7-2 DS2数据程序极简示例

```
proc ds2;
  data d / overwrite=yes;
    declare   char(32) c;
    method init();
      c='ds2 data program';
    end;
  enddata;
run;
quit;

proc print data=d;run;
```

输出结果如图 7-2 所示。

SAS 系统

Obs	c
1	ds2 data program

图 7-2　DS2 数据程序输出到数据集

程序 7-2 与程序 7-3 的传统 DATA 步等价，从中可看出 DATA ... ENDDATA 与 DATA 步的功能类似，不同之处是 DS2 可包括方法定义。

程序7-3 DS2数据程序极简示例对应的 DATA程序

```
data d;
  length c $ 32;
  c='ds2 data program';
run;
proc print data=d;run;
```

DS2 中所有的程序实体必须用标识符预先定义，标识符由字母、数字和下划线组成，且必须以字母或下划线开头，程序实体包括变量、标签、方法、数据程序、包程序、线程程序和数组等。

7.1.1 变量声明与类型

DS2 变量必须用 DECLARE 语句(别名为 DCL) 对变量进行预先声明，且

DECLARE 语句只能出现于程序块的头部或者方法的头部，而不能出现在中间部分。如果违反这一准则，SAS 会报编译错误。DECLARE 语句的完整语法如下：

```
declare 数据类型变量名 having label '文本' format 格式 informat 格式;
```

比如：

```
declare char (8) name; /*定义长度为8字符的定长字符串*/
declare double weight; /*定义Double类型的变量 weight*/
```

由于 DS2 变量可输出到 SAS 数据集中，因此定义时也可指定标签文本以及输入/输出格式信息。

```
declare varchar (32) prdname having label 'product name';
declare varchar (32) prdname2 having label 'product name' format$32.;
```

下面的例子（见程序 7-4）演示了主要数据类型的使用，需要特别注意 VARCHAR 和 CHAR 的不同，VARCHAR 类型的变量尾部不包括自动补齐空格。

程序7-4 DS2 主要数据类型示例

```
proc ds2;
  data _null_ ;
    method init();
      declare  char (8)    c1;  /*定长字符，尾部自动补齐空格*/
      declare  nchar (8)   nc1; /*支持 unicode 字符*/
      declare  varchar(8) vc1;  /*变长字符，尾部不自动补齐空格*/
      declare  nvarchar(8) nvc; /*支持 unicode 字符*/
      declare  double d having label 'DBL' format best. informat best.;
      declare  int i;

      c1 ='abcd'; put c1= '<-';
      nc1 ='唯有变化才是永恒'; put nc1= '<-';

      vc1 ='abcd';  put vc1= '<-';
      vc1 ='abcdefgh123'; put vc1= '<-';

      d=1/3.0; put d=; /*注意小数点，如果写成 1/3 是整数相除，结果等于 0 */
      i=32767; put i=;
    end;
  enddata;
run;
quit;
```

系统输出结果如下：

```
c1=abcd     <
nc1=唯有变化才是永恒 <-
vc1=abcd <-
vc1=abcdefgh <-
d 0.3333333333
i-32767
```

表 7-1 列出了 DS2 的数据类型系统，包括字符、整数、浮点数、日期/时间和二进制 5 大类型，各类型还可细分为 17 种基本数据类型，其中类型 CHAR 和 DOUBLE 对应传统 DATA 步中的字符类型和数值类型。

表 7-1 DS2 的数据类型系统

字符类型 [4]	描 述
CHAR（N）	定长字符型，N 为字符长度； 它与传统 DATA 步中的**字符型变量**，使用 LENGTH $N 语句进行定义等价
NCHAR（N）	定长字符型，N 为字符长度；它与 CHAR（N） 的区别是它支持 UNICODE 多国语言字符；根据平台的不同，每个字符可能需要 2 字节或 4 字节进行存储
VARCHAR（N）	变长字符型，N 为最大字符长度；如果实际字符数小于 N 则变量只存储实际的字符数，并不像前面定长字符型那样总为当初定义的宽度
NVARCHAR（N）	变长字符型，支持 UNICODE 多国语言字符
整数类型 [4]	
TINYINT	相当于 8 位有符号整数，可表示 [−127 ~ 127] 区间的整数； 精度为 3，SAS 内部使用 4 字节存储
SMALLINT	相当于 16 位有符号整数，可表示 [−32,767 ~ 32767] 区间的整数； 精度为 5， SAS 内部使用 5 字节存储
INTEGER 或 INT	相当于 32 位有符号整数，可表示 [−2,147,483,647 ~ 2,147,483,647] 区间的整数； 精度为 10，SAS 内部使用 10 字节存储
BIGINT	相当于 64 位有符号整数，可表示 [−9,223,372,036,854,775,807 ~ 9,223,372,036,854,775,807] 区间的整数； 精度为 19，SAS 内部使用 19 字节存储
浮点数类型 [4]	
REAL	4 字节有符号单精度浮点数，可表示区间为 ± 3.4028235076646E-38 到 ± 3.4028234663852E38 的近似实数，用于存储常规实数。
DOUBLE	8 字节有符号双精度浮点数，可表示区间为 ± 2.22507E-308 到 ± 1.797693E308 的近似实数，用于存储较大实数 它与传统 DATA 步中的**数值型变量**等价
FLOAT（P）	有符号单双精度可变浮点数，若参数 P<25，则是 REAL 型单精度浮点数，否则为 DOBULE 型双精度浮点数，但在 BASE 环境中运行时不可指定 P 值
DECIMAL（P,S）	有符号定点浮点数，用于精确实数表示，而非近似表示。其中精度 P 是指小数点前后总的有效位数，取值区间为 1~52；标度 S 是指小数点后的最大位数，S 必须小于等于 P。比如 DECIMAL （9,2） 表示 9 位有效数字，2 位小数；需要注意在 SAS 代码中用常数给 DECIMAL 赋值只能精确到 14 位。 DECIMAL （P,S）也可写作 NUMERIC （P, S）
日期时间类型 [3]	
DATE	日期型，以格式 YYYY-MM-DD 指定，其中 YYYY 为 0001-9999，MM 为 01-12，DD 为 01-31；比如 1975-09-24
TIME（P）	时间型，以格式 HH：MM：SS[.NNNNNNNNN] 指定，其中 HH 为 00-23，MM 为 00-59，SS 为 00-61（支持闰秒）；N 可选，仅用于高精度计时之用； 若数据源支持，SAS 可用 P 值来指定秒的精度，最大值为 9 表示精确到 10^{9} 秒，即纳秒
TIMESTAMP（P）	日期时间型，以格式 YYYY-MM-DD：HH：MM：SS[.NNNNNNNNN] 指定，为 DATE 和 TIME（P） 类型的组合形式
二进制类型 [2]	
BINARY（N）	定长二进制数据，N 为最大字节长度，N 为必选参数。 如果二进制内容实际长度小于 N，该变量也始终占据 N 个字节长度
VARBINARY（N）	变长二进制数据，N 为最大字节长度，N 并非必选参数。 如果二进制内容实际长度小于 N，该变量只占实际所需的字节长度

上面17种基本数据类型仅在DS2程序运行期间支持，变量一旦持久化写入SAS数据集，SAS数据集仍然只支持传统DATA步的定长字符型和双精度数值型两种基本变量类型。尽管如此，在DS2中访问第三方关系型数据库的数据时，尽可能使用DS2与之对应的数据类型进行交互，避免因系统间数据类型定义不同出现精度损失的问题。

关于缺失值处理，DS2提供两种模式：SAS模式和ANSI模式，它们由PROC DS2过程步选项ANSIMODE控制。默认DS2将不存在的值视为SAS缺失值，但如果启用ANSIMODE，DS2会将不存在的值当作null值。默认的SAS模式将从数据表中读取不存在的值会当作SAS缺失值，而ANSI模式下读入的不存在的值会被转换为null值。如果SAS字符型变量仅包含空格，在SAS模式中该变量值会被当作缺失值看待，但在ANSI模式中该变量就是一个仅包含空格的非空字符串，而不是null值。SAS传统上的特殊缺失值如._、.A和.Z在ANSI模式中都被当作null值看待而不加以区分。考察程序7-5，结果显示只有句号“.”会被当作缺失值看待。

程序7-5 ANSI模式下的缺失值处理
```
proc ds2 ansimode;
  data d / overwrite=yes ;
    declare   char(32) c;
    method init();
      c=' ';
      if missing(c) then put 'blank is missing in sasmode';
      c=.;
      if missing(c) then put 'dot is missing in ansimode';
    end;
  enddata;
run;
quit;
```

7.1.2 程序实体作用域

作用域是一个程序实体可见性或可以访问的作用范围。在传统DATA步中变量是没有作用域概念的：所有的变量在当前步内都是可用的，它们在PDV中也是扁平存储；当引用一个尚未赋值的变量时，变量默认返回缺失值。在SAS宏代码中可以使用%GLOBAL语句和%LOCAL语句来限定宏变量的作用范围，宏函数内用%LOCAL语句来将宏变量定义为局部有效。如果局部变量与全局变量重名，位于宏函数局部符号表中的局部变量具有使全局宏变量不可见的遮蔽效应。

在DS2中，程序实体的作用域要广泛得多，涉及方法、函数、数据、标签、程序变量等。其中：

（1）在某个程序块中，方法外定义的变量为全局变量，同一程序块内的所有方法都可引用它；方法内定义的变量为该方法的局部变量，只能在该方法内引用它。比如程序7-6中init方法和method2方法中的局部变量是相互隔离的。

程序7-6 方法内的变量作用域仅限本方法内
```
proc ds2;
  data _null_ ;
```

```
    declare   int var_global;  /* 数据程序的全局变量*/
    method init();
      declare   int var_local; /* 方法 init 的局部变量1*/
    end;
    method method2();
      declare   int var_local; /* 方法 method2 的局部变量2*/
    end;
  enddata;
run;
quit;
```

（2）数据程序（定义于DATA和ENDDATA语句之间）：该程序块头部定义的变量在数据程序内全局有效，且存活于该数据程序执行期间。如果要在数据程序中输出变量到SAS数据集，该变量必须是该数据程序的全局变量。其中SET语句引用的变量默认具有全局作用域，除非显式调用DROP语句删除变量，否则数据程序中的全局变量会自动包含在数据程序的PDV中。

（3）包程序（定义于PACKAGE和ENDPACKAGE语句之间）：该程序块头部定义的变量在包内具有全局效果。包内的变量不会被自动包括在调用此包实例的数据程序的PDV中，包程序的全局变量仅在DS2包程序实例的存活期间可用。

（4）线程程序（定义于THREAD和ENDTHREAD语句之间）：该程序块头部定义的变量在当前线程程序内全局有效。其中SET语句引用的变量默认具有全局作用域，除非显式调用DROP语句删除变量，否则线程程序内的全局变量会包括在线程输出数据集中。线程程序的全局变量在线程实例执行期间可用，且它们可被传递给数据程序的SET FROM语句。

（5）方法（定义于METHOD-END语句之间）是数据程序、线程程序和包程序的组成部分。方法名默认在该程序块内全局有效。如前所述，方法头部定义的变量为该方法的局部变量，仅存活于该方法调用期间，且不会被包含在数据程序块的PDV中。

7.1.3 变量数组与标准数组

除基本变量外DS2支持两种类型数组：变量数组和标准数组，它们都在DS2程序执行期间可用。其主要区别如下所述。

（1）变量数组使用VARARRAY语句来创建，必须在全局作用域内定义；它与传统DATA步的数组性质完全相同，是全局变量的一种临时分组方式；没有显式指定映射变量系列时，系统会自动创建以数组名和数字命名的一系列变量，如var_array1、var_array2、var_array3等。

（2）标准数组使用DECLARE语句来创建，它既可以在全局作用域中定义，也可以在方法的局部作用域中定义。标准数组的定义包括数组名、数据类型、数组下标以及数组长度等信息，它与传统计算机编程语言C/C++和JAVA中的数组性质完全相同。

下面的代码使用VARARRAY语句创建变量数组（见程序7-7），它会自动在PDV中创建变量VAR_ARRAY1、VAR_ARRAY2和VAR_ARRAY3，这些PDV变量也会自动

输出到结果数据集 MYDATA 中（见图 7-3）。

```
程序7-7  VARARRAY 变量数组示例
proc ds2;
  data mydata(overwrite=yes);
    vararray varchar(16) var_array[3]; /*自动映射变量并输出到数据集*/
    method init();
      var_array: =('Alfred','Leon','Steve');
    end;
  enddata;
run;
quit;
proc print data=mydata;run;
```

Obs	var_array1	var_array2	var_array3
1	Alfred	Leon	Steve

图 7-3　变量数组自动输出到数据集

如果把上面的 VARARRAY 改为 DECLARE 来创建标准数组，则默认输出数据集中不包含任何列，原因就是标准数组并不在 PDV 中创建对应的变量。为了能够将标准数组的元素输出到数据集，需要在数据程序中人为创建对应的全局变量才能输出与前面的例子等价的结果（如程序 7-8）。除此之外两种数组在其他方面没有什么显著区别。

```
程序7-8  标准数组示例
proc ds2;
  data mydata/ overwrite=yes ;
    declare varchar(16) ds2_array[3];
    declare   varchar(16) ds2_array1 ds2_array2 ds2_array3;

    method init();
      ds2_array: =('Alfred','Leon','Steve');

      ds2_array1=ds2_array[1];
      ds2_array2=ds2_array[2];
      ds2_array3=ds2_array[3];
    end;
  enddata;
run;
quit;
proc print data=mydata;run;
```

7.1.4　系统方法与用户自定义方法

DS2 方法是 DS2 程序的主要组成部分，它由一系列可执行语句组成，是仅次于程序块的编程单元。DS2 方法本身在程序块中全局有效，可分为系统预定义方法和用户自定义方法。DS2 方法包含参数与返回值，功能上与 PROC FCMP 中的函数大体相同，其基本语法如下：

```
METHOD METHOD NAME(DATATYPE ARG1,…,DATATYPE ARGN) RETURNS DATATYPE;
  DS2 STATEMENTS;
  RETURN VAR;
END;
```

运行如下样例代码，系统将会调用自定义的方法Add来计算结果并打印（见程序7-9）。

程序7-9 DS2方法示例（以数据程序为例）

```
proc ds2  ;
  data mydata(overwrite=yes);
    declare  double X Y;
    declare  double total having label 'Total X+Y' format 5.1 ;

    method Add (double X, double Y) returns double;
      return (X+Y);
    end;

    method init();
      X=100; Y=200;
      total=Add(X,Y);
    end;
  enddata;
run;
quit;
proc print data=mydata label;run;
```

结果如图7-4所示。

Obs	X	Y	Total X+Y
1	100	200	300.0

图 7-4　DS2 输出数据集

PROC DS2 中定义的方法只能在 PROC DS2 的程序空间内调用，不能在传统 DATA 步中直接调用 PROC DS2 过程步中定义的用户方法。

- *系统方法*

SAS 系统为 DS2 数据程序和线程程序预定义了 4 个系统回调方法，分别是 INIT、RUN、 TERM 和 SETPARMS。这些方法在 DS2 数据程序和线程程序执行期间起作用，它们有固定的调用规则，用户可通过编程覆盖这些系统方法，实现不同阶段完成不同的功能。DS2 包程序不包含这 4 个系统方法。

（1）INIT( ) 方法：初始化方法，没有参数与返回值，在 DS2 程序启动时被自动调用，常用于 DS2 主程序执行前的初始化。

（2）RUN( ) 方法：运行方法，没有参数与返回值，在 DS2 程序的 INIT( ) 方法执行后被调用；默认情况下 DS2 程序从输入表中每读取一行数据就会自动触发调用 RUN 方法一次，这与传统 DATA 步的隐性循环非常类似。通常 DS2 数据程序的 INIT 或 RUN 方法会包含 SET FROM 语句用于指定线程分配，而 DS2 线程程序的 RUN 方法则用来实现并行计算逻辑。

（3）TERM( ) 方法：终止方法，没有参数与返回值，在 DS2 数据程序或线程程序终止时自动调用，通常用于程序结束前的资源清理。注意该方法对程序全局变量的修改默认不会被输出到结果数据集，因为此时系统已经进入终止阶段。

（4）SETPARMS 方法：设置参数方法，用来初始化线程参数，同一线程程序创建出来的多个实例会使用相同的线程参数，这部分将在后面进行详细讲述。

用户也可以重载 4 个系统方法来实现自定义计算逻辑，其中 INIT/RUN/TERM 3 个系统方法都提供执行入口。程序 7-10 重载 DS2 系统方法 INIT() 来实现黄金分割数列生成器，其中数据类型采用的是有符号定点浮点类型 DECIMAL。为了尽可能生成精确表示的整数序列而使用数据类型 DECIMAL（52,0）。

程序7-10　DS2版的斐波那契数列生成器

```
libname mylib "C:\temp";
proc ds2;
  data mylib.fbnc(overwrite=yes);
    declare decimal(52,0) x having label 'fbnc' format best32.;
    method init();
      declare decimal(52,0) n x1 x2;
      do n = 1 to 250;
        if n=1 or n=2 then x=1;
        else x= x1 + x2;

        x2=x1; x1=x;
        output;
      end;
    end;
  enddata;
run;
quit;
proc print data=mylib.fbnc (obs=250);
run;
```

上面的代码尽管使用了 DECIMAL 数据类型，但只能计算数列的前 250 个数值。在计算第 251 个数时会发生计算溢出。其中前 78 个为没有精度损失的结果数列，但从第 79 个数开始出现计算精度损失；系统从第 154 个开始默认使用科学计数法表示。在 250 个相对精确表示的数值中，只有前 16 位数字是相对精确的。不管是利用标准的 DATA 步还是 PROC DS2 步，能够生成的黄金分割数列长度总是有限的。如何编程突破计算机数据类型的存储限制，生成第 78 个以后可以精确表示的黄金分割数列呢？此时需要借助数组和自定义运算逻辑才可实现，可以参考数据结构—数组一章的内容。

为了探索 DS2 各方法的执行时序问题，执行程序 7-11 DS2 数据并检查输出。

程序7-11　DS2数据程序的系统方法执行时序演示

```
proc ds2;
  data d / overwrite=yes;  /*数据程序*/
    dcl int count;
    method init();
      count=1;
      put '> [Data Program' _threadid_ '] INIT';
    end;
    method run();
      count=2;
      put '[Data Program' _threadid_ '] RUN';
    end;
    method term();
      count=3;
      put '[Data Program' _threadid_ '] TERM';
    end;
    enddata;
run;
quit;
proc print data d;run;
```

系统日志输出如下：

```
[Data Program 0 ] INIT
[Data Program 0 ] RUN
[Data Program 0 ] TERM
```

检查发现输出的 SAS 数据集中 count = 2，而不是等于 3。原因是 TERM( ) 执行于数据输出之后，因此 TERM( ) 方法内执行 count=3 并不会默认输出到数据集中。但是可以在 TERM 方法中显式调用 output 语句将 PDV 中的变量输出到数据集。

DS2 线程程序有相同类似的执行时序，但启动线程需要在 DS2 数据程序中创建线程实例触发线程执行。下面的例子综合展示了线程程序的执行时序，其中 DS2 线程程序实例的 SETPARMS 方法可以设置线程参数，而 SET FROM 语句可以创建多个线程实例，各线程实例是并行执行的。考察如下例子，它创建了线程程序 T 的 3 个实例（见程序 7-12）。

程序7-12　DS2线程程序的系统方法执行时序演示

```
proc ds2;
  /*线程程序*/
  thread t( varchar(20) param1 ) / overwrite=yes;
    dcl int threadid_child;
    method init();
      put '  [Thread Program' _threadid_ '] INIT''(param1=' param1 ')';
      threadid_child=_threadid_;
    end;
    method run();
      sleep(1000);/*模拟执行任务*/
      put '  [Thread Program' _threadid_ '] RUN';
    end;
    method term();
      put '  [Thread Program' _threadid_ '] TERM';
    end;
  endthread;

  data d / overwrite=yes; /*演示程序*/
    dcl thread t t1;
    dcl int count;
    method init();
      put '[Data Program' _threadid_ '] BEGIN';
      t1.setparms('WORKER' );/*设置线程参数*/
      set from t1 threads=3;/*创建 3 个子线程*/
      put '[Data Program' _threadid_ '] END';
    end;
  enddata;
run;
quit;
```

执行后系统输出如下，说明各线程是并行执行，但对于每个线程还是按照 INIT->RUN->TERM 的时序执行。

```
[Data Program 0 ] BEGIN
  [Thread Program 1 ] INIT (param1= WORKER )
  [Thread Program 2 ] INIT (param1= WORKER )
  [Thread Program 3 ] INIT (param1= WORKER )
  [Thread Program 3 ] RUN
```

```
    [Thread Program 2 ] RUN
    [Thread Program 1 ] RUN
    [Thread Program 3 ] TERM
    [Thread Program 1 ] TERM
    [Thread Program 2 ] TERM
  [Data Program 0 ] END
```

- 自定义方法

3 种 DS2 程序都支持创建用户自定义方法。方法可以包含参数和返回值，且方法参数要求声明数据类型。DS2 也支持方法重载——即用相同的方法名，但根据参数类型列表不同来定义不同的方法。在运行的时候，SAS 会根据实际参数的个数、类型和顺序的不同自动调用与其对应的 DS2 方法。

如果参数个数匹配，但参数类型不匹配，DS2 会试图对实际参数进行自动类型转换来调用函数。但对于 DATE、TIME、TIMESTAMP、VARBINARY4 种数据类型，系统不做默认的类型转换，实际参数必须为形式参数指定的数据类型。如果找不到匹配参数个数的方法，SAS 会抛出调用错误。

DS2 方法参数在定义时还可使用 IN_OUT 修饰符进行修饰，表示该参数是否为输入输出参数。输入，输出参数在方法调用结束后，方法中对参数的修改会被传回调用方。这一机制为我们在单个方法中返回多个数据或者复杂数据交换提供了机制，类似于 FCMP 函数参数的 OUTARGS 语句功能。

DS2 方法可以没有返回值，但如果在方法定义时使用了 RETURNS 指定返回值类型，则必须在方法体内使用 RETURN 语句返回一个值。在绝大多数编程语言中，方法返回值并不是方法签名的组成部分，因此不要试图使用定义名称和参数类型列表相同，但返回值不同的方法。

下面的例子展示了如何在方法中用返回值表示计算是否成功，如两个数相除时要求分母不得为 0，就可设计返回一个标志量（见程序 7-13）。

程序7-13 DS2 方法返回标志量，利用输出参数返回计算结果示例

```
proc ds2;
  data _null_ / overwrite=yes;
    method Div(double x, double y, in_out double result) returns int;
      if y=0 then return 0; /*返回0 表示失败-除数为零不能相除*/
      result=x /y;          /*相除结果在 result 中*/
      return 1;             /*返回1 表示成功*/
    end;
    method run();
      declare double res;
      declare int rc;

      rc=Div(3, 4, res);
      if rc=1 then put res=;
      else put 'ERROR: 发生分母为 0 错误! ';

      rc=Div(3, 0, res);
      if rc=1 then put res=;
      else put 'ERROR: 发生分母为 0 错误! ';
    end;
  enddata;
```

```
run;
quit;
```

当DS2方法跟SAS系统函数重名时，DS2方法在DS2程序中具有优先级，而SAS系统函数相当于被遮蔽起来。此时如果要调用SAS系统函数而非同名的DS2函数，需要使用SYSTEM包前缀来指明调用的目标为SAS系统函数。比如下面的例子中，自定义DS2方法DATE与系统函数DATE重名，就需要明确使用SYSTEM包前缀（见程序7-14）。

程序7-14 函数隐藏问题与解决办法

```
proc ds2;
  data _null_;
    method date();      /*与SAS 系统函数 DATE() 重名*/
      declare   int d;
      d=SYSTEM.date(); /*调用系统函数需要带上包名 SYSTEM*/
      put d= date9.;
    end;
    method run();
      date();
    end;
  enddata;
run;
quit;
```

7.2 数据程序

正如前面一些例子所揭示的，DS2中的数据程序继承了传统DATA步的功能，但语法上它必须存在于PROC DS2的DATA…ENDDATA之间。主要区别是DS2数据程序具有上面讲到的变量作用域和系统预定义方法INIT、RUN和TERM等。其中，RUN方法用于和实现传统DATA步类似的隐形循环机制。语法上DS2看起来比传统DATA步更烦琐，但它提供更强大而精细的控制。下面计算体质指数的例子（见程序7-15）展示了DS2数据程序的基本工作机理，注意隐性循环的作用。

程序7-15 DS2版的隐性循环与生成计算列

```
data class;
  set sashelp.class;
run;

proc ds2;
  data d / overwrite=yes;
    dcl double bmi;
    method run();
      set class
      bmi=(weight * 0.4535924) / (height * 0.0254)**2;
    end;
  enddata;
run;
quit;
proc print data d;run;
```

需要注意的是，对于运行在 IN-DATABASE 库内计算环境的 DS2 数据程序，SAS 9.4 中非 SET FROM 语句是在 SAS 里执行的，线程程序的输出是从数据库抽取数据到 SAS 环境中；而 SAS 9.4M1 为充分利用计算环境的并行处理机制，减少数据传输负载，SAS 将数据程序内的代码部署运行在一个数据库节点的并行处理单元，如 Teradata 的 AMP 上。此时，线程程序是通过一个数据库临时表给数据程序传回数据。

7.3　包程序

DS2 包程序中的包（PACKAGE）概念类似于传统面向对象编程中的类（CLASS），用于将变量和方法进行封装打包，形成一个可重复使用的代码复用单元。DS2 包可以持久化到磁盘上，然后在另一个 SAS 程序中调用它。由于 DS2 包在数据分析领域的特殊性，它并不需要支持继承。

下面的 DS2 包程序定义了一个 Person 包，它相当于面向对象编程中的一个类（见程序 7-16）。

程序7-16　DS2版的 Person类

```
proc ds2; /*定义 Person 类*/
  package Person / overwrite=yes;
    declare nvarchar(32) name sex;
    method setname( nvarchar(32) n);
      name=n;
    end;
    method getsex( ) returns nvarchar(32);
      return sex;
    end;
    method eat();
      put name '吃饭';
    end;
    method sleep();
      put  name  '睡觉';
    end;
    method work();
      put  name  '工作';
    end;
    method think(double x, double y);
      declare double z;
      z=sqrt(x*x+y*y);
      put  name  '思考：勾' x '股' y '=>弦' z;
    end;
  endpackage;
run;
quit;
```

DS2 包变量的声明与 DS2 变量声明类似，只需要将数据类型改为 PACKAGE XXX 即可。但包变量还需要使用 new 关键字进行实例化，实例化后的包对象就可以使用对象成员访问符“.”号来访问成员。下面的例子演示了 Person 包的定义和实例化过程（见程序 7-17）：

程序7-17　DS2版的 Person类的实例化与调用示例

```
proc ds2;/*测试 Person 类*/
  data _null_ / overwrite=yes;
    declare nvarchar(20) name sex;
    method init();
      declare package Person p;/*定义变量 p*/
      p=_new_ Person(); /*实例化变量 p*/
      p.setname('里昂');
      p.sex='男';
      p.eat();
      p.sleep();
      p.work();
      p.think(3,4);

      name=p.name;
      sex=p.getsex();
    end;
  enddata;
run;
quit;
```

系统输出：

```
里昂 吃饭
里昂 睡觉
里昂 工作
里昂 思考：勾 3 股 4 =>弦 5
```

包实例的作用域和DS2变量的作用域规则相同，DS2包实例也可作为DS2方法的参数或返回值进行传递。创建一个DS2包变量可以采用如下两种方式之一。

（1）先定义然后用 _new_ 关键字实例化，实例化时可使用 [this] 修饰来指定对象全局有效。

```
declare package mypack o; /*先定义，后实例化*/
o=_new_ mypack();
```

或

```
o=_new_ [this] mypack();  /*先定义，后实例化且使之全局有效*/
```

（2）定义时直接实例化，与DS2变量定义的区别就是需要加上括号。

```
declare package mypack o();/*定义的时候直接实例化*/
```

与传统的OOP不同，不要试图在某个包的成员方法中实例化该包自身，否则会报编译错误“Cannot instantiate package ... from within itself.”，但可以实例化非自身包对象；DS2也不允许在包方法中试图返回当前实例的引用this。

DS2包程序跟DS2数据程序和线程程序的不同之处在于，它没有3个系统预定义方法：INIT( )、RUN( )和TERM( )。DS2包程序主要是用来实现自定义函数库和代码重用。下面的DS2示例实现了复数类以及有关运算（见程序7-18）。

程序7-18　DS2版的“复数类”的实现

```
proc ds2;/*复数类*/
  package Complex / overwrite=yes;
```

```
declare   double a b;

method init(double a, double b);/*初始化复数，类似于构造函数*/
  this.a=a; this.b=b;
end;

method print(); /*打印复数*/
  put  a '+' b 'i';
end;

method norm() returns double; /*计算复数的模，即复数的绝对值*/
  return sqrt(this.a ** 2 + this.b ** 2);
end;

method add(double a, double b);     /*复数四则运算封装：  基本类型*/
  this.a = this.a + a;
  this.b = this.b + b;
end;
method sub(double a, double b);
  this.a = this.a - a;
  this.b = this.b - b;
end;
method mult(double a, double b);
  declare double aa bb;
  aa = this.a * a - this.b * b;
  bb = this.a * b + a * this.b;
  this.a=aa; this.b=bb;
end;
method div(double a, double b);
  declare double sum aa bb;
  sum= a ** 2 + b ** 2;
  aa = (this.a * a + this.b * b) / sum;
  bb = (this.b * a - this.a * b) / sum;
  this.a=aa; this.b=bb;
end;

method init(package Complex c); /*复数四则运算封装：  复数类型*/
  this.a=c.a; this.b=c.b;
end;
method print(package Complex c); /*重载方法 print()*/
  declare double a b;
  a=c.a; b=c.b;
  put  a '+' b 'i';
end;
method add(package Complex c); /*重载方法 add(double a, double b)*/
  this.a = this.a + c.a;
  this.b = this.b + c.b;
end;
method sub(package Complex c);
  this.a = this.a - c.a;
  this.b = this.b - c.b;
end;
method mult(package Complex c);
  dcl double aa bb;
  aa = this.a * c.a - this.b * c.b;
  bb = this.b * c.a + this.a * c.b;
  this.a=aa; this.b=bb;
end;
method div(package Complex c);
  declare double aa bb sum;
  sum= c.a ** 2 + c.b ** 2;
```

```
        aa= (this.a * c.a + this.b * c.b) / sum;
        bb = (this.b * c.a - this.a * c.b) / sum;
        this.a = aa; this.b = bb;
      end;
    endpackage;
run;
quit;
```

程序 7-19 则是该复数类的应用示例：

程序7-19 DS2复数类的应用示例

```
proc ds2;
  data _null_;
    method init();/*测试 Complex 类*/
      declare package Complex c1();
      declare package Complex c2();
      declare double n;
      c1.a=3; c1.b=4; c1.print();
      n = c1.norm(); put 'norm=' n; put '';

      c2.init(6, 8); c2.print();
      n = c2.norm(); put 'norm=' n; put '';

      put 'Complex add';
      c2.add( c1); c2.print();
      n = c2.norm(); put 'norm=' n;  put '';

      put 'Complex mult';
      c1.a=0; c1.b=1; c1.print();
      c1.mult(c1);  c1.print();
      n=c1.norm(); put 'norm=' n; put '';
    end;
  enddata;
run;
quit;
```

系统输出：

```
3 + 4 i
norm= 5

6 + 8 i
norm= 10

Complex add
9 + 12 i
norm= 15

Complex mult
0 + 1 i
-1 + 0 i
norm= 1
```

● 集成性

在 DS2 代码中可以调用自定义的 FCMP 函数，但调用时需要使用包程序创建一个代码封装器，然后再通过该代码封装器调用对应的 FCMP 函数。完整的例子见程序 7-20。

程序7-20　待封装的自定义 FCMP函数示例

```
libname mylib "C:\temp";
proc fcmp outlib=mylib.Funcs.math;
  function add(arg1, arg2); /*自定义 FCMP 加法函数*/
    ret=arg1+arg2;
    return(ret);
  endsub;
run;
```

创建FCMP函数库MYLIB.FUNCS完后，如果需要在传统的DATA步中调用该方法，一般使用OPTIONS CMPLIB=WORK.FUNCS;选项来指定CMPLIB即可。但在DS2中，需要使用特殊的包程序来桥接FCMP代码（见程序7-21）。

程序7-21　将编译的FCMP 函数包封装为一个特殊的DS2 包程序

```
proc ds2;
  package mypack/overwrite=yes language='fcmp' table='mylib.Funcs';
  endpackage;
run;quit;
```

这样就形成了一个特殊的DS2包mypack，然后就可以像调用其他DS2包一样调用FCMP方法（见程序7-22）。

程序7-22　通过DS2 包程序调用自定义FCMP 函数实现

```
proc ds2;
  data;
    declare double r;
    method init();
      declare package mypack o(); /*与其他 DS2 函数调用方式没有差异*/
      r=o.add(100, 200);
      put r=;
    end;
  enddata;
run;
quit;
```

DS2除了可以调用FCMP函数和例程外，还可通过如下系统预定义包来调用系统函数。

（1）HASH/HITER包：提供基于键值对的高效数据存储和查找的哈希表功能。

（2）MATRIX包：提供强大灵活矩阵编程访问接口，它不要求SAS/IML软件授权即可在SAS中做矩阵运算。

（3）SQLSTMT包：能够利用FedSQL语句将SQL传入关系型数据库执行，并访问关系数据库查询返回的结果集。

（4）HTTP包：构造访问HTTP Web服务的HTTP客户端

（5）JSON包：用于创建和解析JSON文本数据

（6）LOGGER包：访问SAS日志基础设施的基本接口，包括读/写/查找等。

（7）TZ包：提供国家区域Locale和国际化时间/日期值有关的处理。

其中HASH/HITER和MATRIX包在分析实践中比较常用，其中HASH/HITER对象在传统的DATA步中也可使用，SAS中的哈希对象以及遍历器在数据查找变换中非常有用，它支持多键值和多数据查找。下面的例子（见程序7-23）在DS2中创建联合键值的哈希对象，其数据分别为键值的和与积，然后查找某个联合键值对应的数据。需特别注

意 SAS 的哈希对象可以包含复合键值，且可有多个值对象。

程序7-23 DS2Hash/HIter 对象使用示例

```
proc ds2;
  data _null_ ;
    declare double k1 k2 d1 d2 ;
    dcl package hash h();
    dcl package hiter hi(h);/*关联遍历器*/

    method init();
      dcl int rc;
      /*定义联合键值的哈希表 [k1, k2]={d1, d2} */
      h.defineKey('k1'); h.defineKey('k2' );
      h.defineData('d1' ); h.defineData('d2');
      h.defineDone();

      /*初始化 Hash 表，其中 d1=k1+k2, d2=k1*k2 */
      put 'Init Hash Object';
      do k1 = 1 to 3;
        do k2 = 1 to 3;
          d1 = k1+k2;
          d2 = k1*k2;
          h.add();
        end;
      end;

      /*用 HIter 遍历哈希表，打印出来*/
      rc = hi.first();
      do while(rc = 0);
        put d1= d2=;
        rc = hi.next();
      end;

      /*创建联合键值 [2, 3] 并查找*/
      k1=2; k2=3;
      h.find();
      put'[' k1 ',' k2 ']={' d1 ',' d2 '}';
    end;
  enddata;
run;quit;
```

系统最后输出 [2 , 3]={ 5 , 6 }

- 实例

下面的例子（见程序 7-24）展示了如何在 SAS 中利用矩阵运算包 MATRIX 实现矩阵乘法运算，它是线性代数运算和方程求解的基础。使用 DS2 矩阵运算包不需要 SAS/IML 专业矩阵运算包的软件授权。该例子执行如下计算：

$$C=AB=\begin{bmatrix}1&2&3\\4&5&6\end{bmatrix}\begin{bmatrix}1&4\\2&5\\3&6\end{bmatrix}=\begin{bmatrix}1\times1+2\times2+3\times3&1\times4+2\times5+3\times6\\4\times1+5\times2+6\times3&4\times4+5\times5+6\times6\end{bmatrix}=\begin{bmatrix}14&32\\32&77\end{bmatrix}$$

程序7-24 DS2矩阵对象使用示例

```
proc ds2;
  data _null_ ;
    /*将数组格式化打印输出*/
    method print(double x[*], int r, int c);
```

```
    dcl int i j;
    dcl nvarchar(32) line;

    do i=0 to r-1;
      line='';
      do j=1 to c ;
        if j>1 then line=line || ', ';
        line=line || put( x[i*c + j], 3.);
      end;
      put '[' line ']';
    end;
  end;

  /*程序入口：矩阵乘法运算*/
  method init();
    dcl double aa[2,3] bb[3,2];
    dcl package Matrix A  B C;
    dcl double cc[2,2];

    aa: =(1,2,3, 4,5,6);
    A=_new_ Matrix(2,3); A.load(aa);

    bb: =(1, 4, 2, 5, 3, 6);
    B=_new_ Matrix(3,2); B.load(bb);
    C= A.mult(B);

    C.toArray(cc);
    put 'Atrix A'; print(aa, dim(aa,1), dim(aa,2) );
    put 'Atrix B'; print(bb, dim(bb,1), dim(bb,2) );
    put 'Atrix C'; print(cc, dim(cc,1), dim(cc,2) );

  end;
enddata;
run;
quit;
```

系统输出：

```
Matrix A
[   1,    2,    3 ]
[   4,    5,    6 ]
Matrix B
[   1,    4 ]
[   2,    5 ]
[   3,    6 ]
Matrix C
[  14,   32 ]
[  32,   77 ]
```

7.4　线程程序

DS2 通过线程程序（THREAD... ENDTHREAD）实现多线程支持，它能实现对数据表中的不同观测，并发执行用户定义的处理逻辑。在传统单机环境中 SAS 是以单进程方式执行的，后来随着引入 THREAD KERNEL 技术（简称 TK 技术），SAS 才支持多用户服务器模式进行多线程并发处理。

由于SAS是专门面向数据分析而设计的高级语言，因此DS2多线程的设计思路依然是围绕数据分析展开。DS2为我们封装了线程处理的底层细节，SAS程序员只需要考虑如何分区数据，处理数据即可。DS2线程程序有两种处理模式：

（1）作为程序运行：输入数据为来自数据库表的记录或者DS2程序生成的记录；输出数据为数据库表，或者返回主调程序的数据行。

（2）作为线程运行：输入数据为来自数据库表的记录，每个线程读取数据然后执行必要的取子集、求值等操作然后将结果返回主调程序。

下面的DS2代码定义了一个最简单的线程程序（见程序7-25），并创建3个线程实例并行执行。

程序7-25 DS2线程程序创建3个实例

```
proc ds2;
  thread t / overwrite=yes;
    declare   char(32) c;
    method init();
      put 'Hello DS2 Thread' _threadid_;
    end;
  endthread;
run;
quit;

proc ds2;
  data _null_;
    declare thread t t1;
    method init();
      set from t1 threads=3;
    end;
  enddata;
run;
quit;
```

其输出结果为

```
Hello DS2 Thread 0
Hello DS2 Thread 1
Hello DS2 Thread 2
```

为了说明SAS多线程处理机制，下面举一个完整的例子来说明SAS是如何实现海量数据并行计算的（见程序7-26）。首先使用传统的DATA步模拟生成2017年最新的除港澳台外的中国大陆总人口13.5亿人的年龄数据，存入SAS数据集文件D，系统耗时约61秒，生成的文件约10Gb。

程序7-26 生成 1~100 均匀分布的13亿人的年龄数据

```
data d;
  do i = 1 to 1355692576;
    x= ceil(rand("UNIFORM")* 100); /*为简化假定为均匀分布，实际应为指数分布*/
    output;
  end;
  drop i;
run;
```

如果采用等价的DS2多线程程序在单机上生成测试数据，系统耗时将是1.7倍。原因是在对称多处理器SMP环境上，多个线程对单个文件进行写操作并不能改进性能，

反而可能因线程调度和文件写冲突而降低性能。因此要根据计算架构的实际需求来编写 SAS 多线程程序。在分布式海量并行 MPP 计算环境上，由于没有文件写冲突问题和共享锁问题，因此生成分布式分区文件会比 SMP 架构性能好得多。

DS2 多线程在读取数据进行分析方面具有巨大的优势。比如需要统计全国 13.5 亿人的平均年龄，可采用如下传统的 DATA 步程序实现（见程序 7-27）。

程序7-27　用DATA 步计算平均年龄

```
data c;
  set d end=last;
  sum+x;
  if last then do;
    avg=sum/_N_;
    output;
  end;
  keep sum avg;
run;
```

在普通单机环境上运行该程序计算 13.5 亿人的年龄总和为 68463510554 岁，平均年龄为 50.50 岁。系统处理实际耗时为 40.28 秒，CPU 时间为 38.59 秒。CPU 时间为程序指令所消耗的总 CPU 时钟数，是计算负载的指示，在多核环境下它与实际时间可能不同。如果用 PROC MEANS 过程步进行计算（见程序 7-28），发现系统实际耗时 44.67 秒，跟上面的程序差不多，但 CPU 时钟消耗为 2 分 18.53 秒耗时有显明增长，这可能与 SAS 过程步 PROC MEANS 的具体实现方式有关。

程序7-28　用 PROC MEANS 计算平均年龄

```
proc means data=d sum mean;
run;
```

下面编写多线程 DS2 程序（见程序 7-29）来完成同样的功能，首先用 1 个线程模式运行，理论上耗时应该与 DATA 步等价。

程序7-29　用DS2 线程程序计算平均年龄（线程数指定为 1）

```
proc ds2;
  thread thread_a / overwrite=yes;
    dcl double id cnt sum;
    keep id cnt sum;
    method run();
      set d;
      sum + x; /*线程体执行局部数据加总*/
    end;
    method term();
      id = _threadid_; /*线程编号，0 到 n-1 */
      cnt = _N_-1;      /*当前线程处理的记录数*/
      output;
      put '线程 [' _threadid_ '] 数据=' cnt '合计=' sum;
    end;
  endthread;
run;
quit;

proc ds2;
  data c2 / overwrite=yes;
    dcl thread thread_a t();
```

```
    dcl double cnt_all avg_all sum_all;
    keep cnt_all sum_all avg_all;
    method run();
      set from t threads=1; /*指定线程数量为 1，用1个线程运行该程序*/
      sum_all + sum;
      cnt_all + cnt;
    end;
    method term();
      avg_all = sum_all / cnt_all;
      output;
      put '线程*[' _threadid_ '] 数据=' cnt_all '合计=' sum_all '
        均值=' avg_all;
    end;
  enddata;
run;
quit;
```

系统输出日志如图 7-5 所示。

```
线程 [ 1 ] 数据= 1355692576 合计= 68460496318
线程*[ 0 ] 数据= 1355692576 合计= 68460496318 均值= 50.4985403991767
NOTE: Execution succeeded. One row affected.
3507  quit;

NOTE: "PROCEDURE DS2" 所用时间（总处理时间）:
      实际时间          39.08 秒
      CPU 时间          50.40 秒
```

图 7-5　单个线程执行结果

运行实际时间为 39.08 秒，CPU 时间为 50.40 秒。单线程处理 13 亿行数据与前面的传统 DATA 步程序 40.28s 差不多，而 CPU 时间从 38.59 秒增大到 52.04 秒，说明多线程程序增大了线程资源的开销时间。如果我们把上面程序的线程数提高到 4 或更高，如 5 个线程时，则该程序在同样的环境上实际消耗时间将会明显缩短到约 13 秒（见图 7-6）。

```
线程 [ 3 ] 数据= 277553152 合计= 14015803394
线程 [ 5 ] 数据= 267304960 合计= 13498310777
线程 [ 1 ] 数据= 268779520 合计= 13573377410
线程 [ 4 ] 数据= 266231808 合计= 13444819752
线程 [ 2 ] 数据= 275823136 合计= 13928184985
线程*[ 0 ] 数据= 1355692576 合计= 68460496318 均值= 50.4985403991767
NOTE: Execution succeeded. One row affected.
3601  quit;

NOTE: "PROCEDURE DS2" 所用时间（总处理时间）:
      实际时间          13.29 秒
      CPU 时间          59.09 秒
```

图 7-6　五个并发线程执行结果

从结果可以看出，DS2 程序 5 个线程并行执行使运行的实际时间从 40.64 秒缩短到 13.29 秒，速度提高了 3 倍。这还仅仅是在普通台式机上 BASE 环境中的表现，在分布式并行计算环境 MPP 上，如 SAS 库内计算或 SAS 内存计算环境上，其并发性能将随着节点规模的增大而提升，直至将对整个 10 亿行数据的计算时间控制在 1 秒以内。

通常在 SMP 环境上 DS2 的线程数并不是越高越好，而是要根据具体的计算环境和数据负载情况合理设置，通常在 4~8 个线程数即可获得非常好的性能提升。图 7-7 为笔

者在普通 SMP 环境上运行程序所消耗的实际时间 / CPU 时间和并发线程数（横坐标）的关系图。

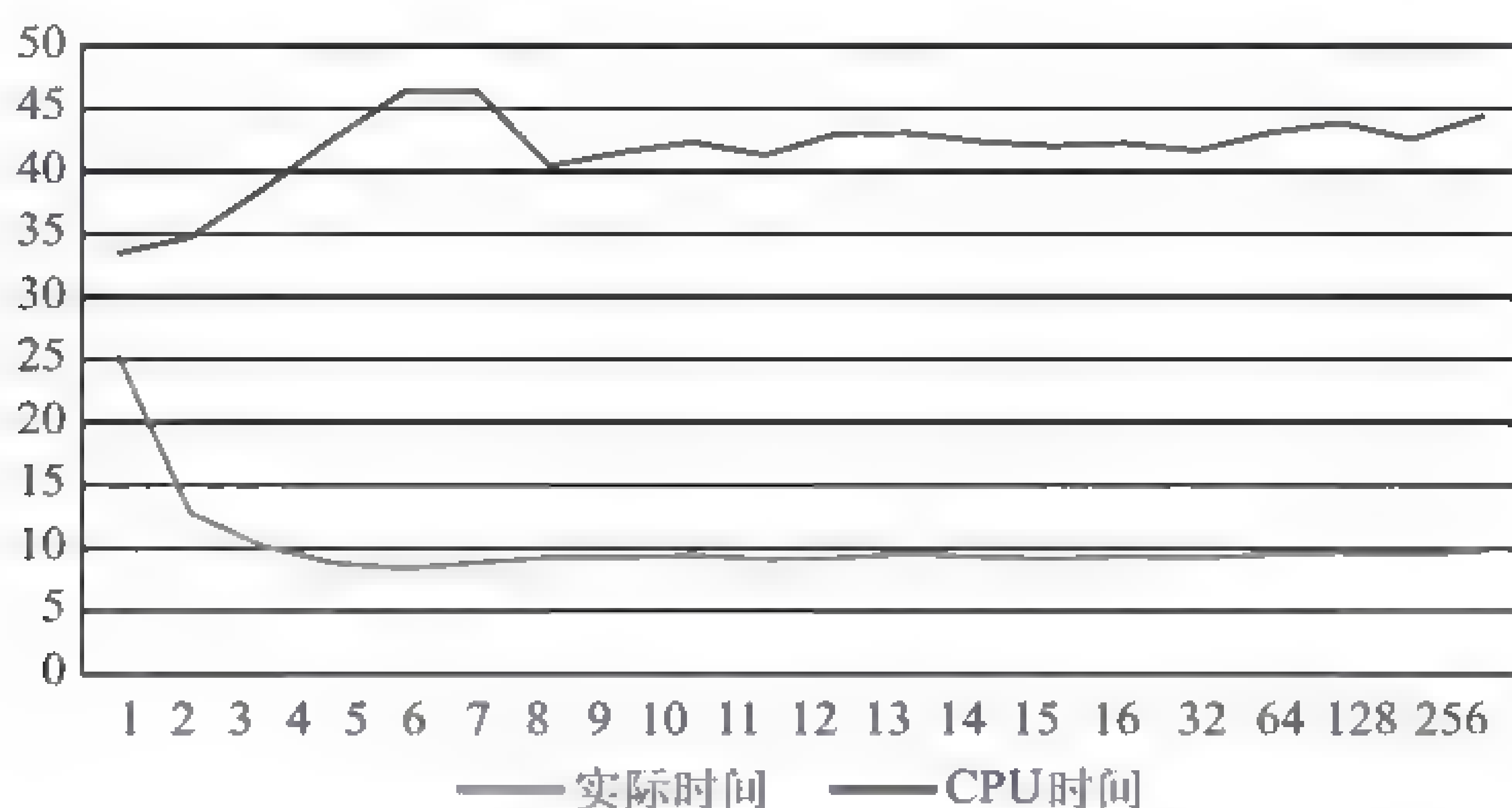


图 7-7　实际时间 / CPU 时间和并发线程数的关系

默认情况下，如果需要处理的数据行数比较少，即使指定了 DS2 的并行处理线程数，SAS 也会使用单个线程进行处理。经笔者验证，实际上如果观测数（记录行数）小于 16384（即 16k）时，则 SAS 通常不会真正启动多线程去执行，因为线程资源本身也是需要额外开销的。

如果我们希望自己控制每个线程的计算负载，也就是说按照用户希望的方式让每个线程处理自己应该处理的那部分数据，如线程 1 处理男性的数据，线程 2 处理女性的数据。下面以 SASHELP.CLASS 数据为例，笔者设计了两个线程分别处理男女性别的数据（见程序 7-30），实践中对于海量数据的任务划分会更加复杂。

程序7-30　自己控制线程负载示例

```
data myclass;
  set sashelp.class;
run;
proc ds2;
  thread thread_a / overwrite=yes;
    dcl double cnt sum avg;
    keep cnt sum;
    method init();
      put '线程' _threadid_ '启动';
    end;
    method run();
      set myclass;
      by sex; /*按照 myclass 中的 sex 变量分区*/
      sum + age;
      put '线程' _threadid_ '处理 ' sex= age= name=;
    end;
    method term();
      cnt = _N_-1;
      if cnt>0 then avg = sum / cnt;
      output;
      put '线程' _threadid_ '结束 ' cnt= sum= avg= 4.1;
    end;
  endthread;
run;
quit;
```

数据和线程程序都已经准备完毕，我们可以编写如下数据程序（见程序 7-31）启动两个线程实例进行处理，线程处理结果的合并是在数据程序的 TERM( ) 方法中进行的。

程序7-31　创建两个线程实例进行处理，结果输出到c2 数据集中

```
proc ds2;
  data c2 / overwrite=yes;
    dcl thread thread_a t();
    dcl double cnt_all avg_all sum_all;
    keep cnt_all sum_all avg_all;
    method run();
      set from t threads=2;/*为每个sex分配一个线程*/
      sum_all + sum;
      cnt_all + cnt;
    end;
    method term();
      avg_all = sum_all / cnt_all;
      output;
      put '线程' _threadid_ '结束 观测=' cnt_all '合计=' sum_all '均值
        =' avg_all 4.1;
    end;
  enddata;
run;
quit;
```

输出结果如图 7-8 所示。

```
线程 1 启动
线程 2 启动
线程 2 处理  Sex=女    Age=12 Name=简
线程 1 处理  Sex=男    Age=12 Name=詹姆斯
线程 2 处理  Sex=女    Age=15 Name=玛丽
线程 1 处理  Sex=男    Age=12 Name=约翰
线程 2 处理  Sex=女    Age=13 Name=芭芭拉
线程 1 处理  Sex=男    Age=16 Name=菲利普
线程 2 处理  Sex=女    Age=12 Name=罗伊斯
线程 1 处理  Sex=男    Age=15 Name=罗纳德
线程 2 处理  Sex=女    Age=14 Name=朱迪
线程 1 处理  Sex=男    Age=11 Name=托马斯
线程 2 处理  Sex=女    Age=11 Name=乔伊斯
线程 1 处理  Sex=男    Age=13 Name=杰弗瑞
线程 2 处理  Sex=女    Age=13 Name=爱丽丝
线程 1 处理  Sex=男    Age=14 Name=亨利
线程 2 处理  Sex=女    Age=14 Name=凯露
线程 1 处理  Sex=男    Age=15 Name=威廉
线程 2 处理  Sex=女    Age=15 Name=雅妮特
线程 1 处理  Sex=男    Age=14 Name=阿尔弗雷德
线程 1 处理  Sex=男    Age=12 Name=罗伯特
线程 2 结束  cnt=9 sum=119 avg=13.2
线程 1 结束  cnt=10 sum=134 avg=13.4
线程 0 结束 观测= 19 合计= 253 均值= 13.3
NOTE: Execution succeeded. One row affected.
3715  quit;

NOTE: “PROCEDURE DS2”所用时间（总处理时间）:
      实际时间          0.44 秒
      CPU 时间          0.34 秒
```

图 7-8　自定义控制线程负载输出

从上面的例子可以看出 SAS 为多线程环境下自定义计算负载提供了灵活的控制接口，合理利用 BY 变量或数据分区，DS2 线程程序能够优化各个线程的计算负载，从而合理分配并行计算负载。根据木桶原理，执行最慢的那个线程实例是并行计算中优化总计算时间的关键。

第 8 章

代码组织

大型分析项目需要多人通力合作或者分模块进行编写，这样才符合化整为零，分解复杂问题然后各个击破的思路。在 SAS 中代码复用可以是源代码文件的静态复用，也可以是前面讲到的 FCMP 函数库或 SAS 宏函数复用。其中静态代码复用主要通过系统宏 %INCLUDE 实现。

8.1 静态文件包含

系统宏 %INCLUDE 用于告诉 SAS 编译器将整个外部文件的内容包含到当前代码中，与当前程序代码一起编译执行。编译器会自动将该文件的内容当作当前代码文件的一部分，放到缓冲区中供后续编译运行使用。其中 ENCODING 选项可用于指明代码文件编码，其基本语法如下。

```
%INCLUDE source </<ENCODING='encoding-value'> <host-options> > ;
```

假如我们编写了一个准备数据的源文件 C:\temp\initdata.sas 如下（见程序 8-1）：

程序8-1 准备数据的SAS程序

```
data mydata;
  input x @@;
  infile datalines;
  datalines;
1 3 5 7 9 2 4
6 8 10
run;
```

这时我们需要在另一个 SAS 程序 printdata.sas（见程序 8-2）里使用同样代码来准备数据，则我们可以使用 %INCLUDE（可缩写为 %INC）来直接包含它，宛如那些代码就是当前文件的一部分，这样程序员就不必复制代码到当前源文件中来。

程序8-2 包含外部源代码示例

```
%include "C:\temp\initdata.sas";
proc print data=mydata;run;
```

程序 8-2 输出结果如图 8-1 所示。

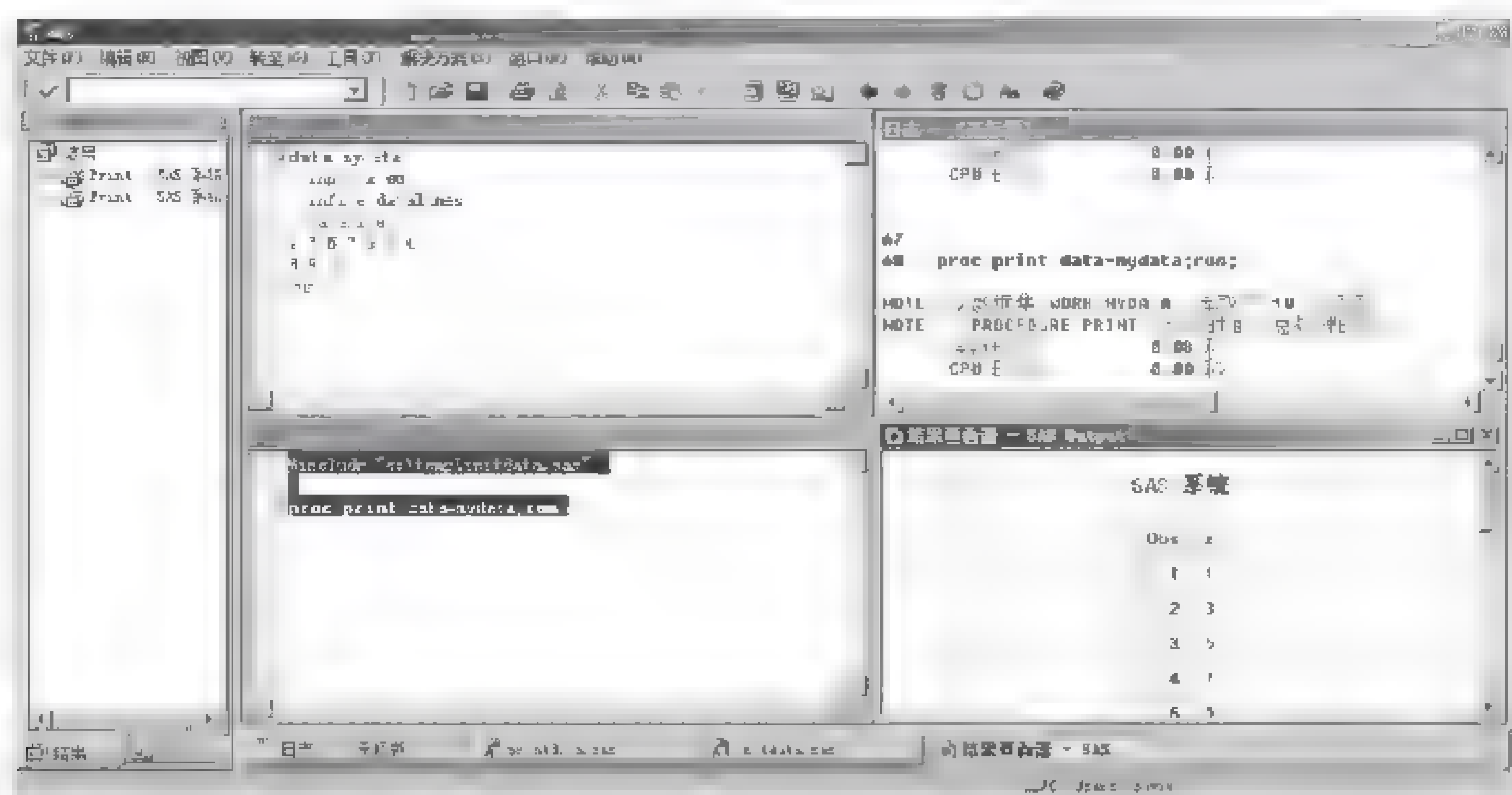


图 8-1 %INCLUDE 代码包含

然而，SAS 里的 %INCLUDE 宏与 C/C++ 的预处理宏 #include 不同，其主要特征如下：

（1）%INCLUDE 语句是一个 SAS 系统宏，它可以出现在源文件中除了 DATA 步 DATALINES 或 DATALINES4 语句后的数据行以外的任何位置。

（2）%INCLUDE 语句所包含的外部源代码文件中，也可以出现 %INCLUDE 语句包含另外一个源代码文件。这种包含机制可以实现多层次的树状 SAS 代码组织，但应避免循环包含。

（3）%INCLUDE 语句是全局的非执行语句，所以不要企图在 DATA 步的条件分支逻辑中使用 %INCLUDE 语句来实现条件包含，只能在 SAS 宏中用 %IF 语句实现条件包含。

我们也可以采用 filename 语句预先定义文件引用，然后再用 %INCLUDE 宏包含该文件引用来包含文件（见程序 8-3）。

程序8-3 包含用filename 指向的源代码示例

```
filename initdata "C:\temp\initdata.sas"
%include initdata;
proc print data=mydata;run;
```

默认情况下并不会显示被包含文件的源代码，如果我们需要在 SAS 日志中显示被包含文件的源代码，可使用 /source2 选项启用（见程序 8-4）；然而打开此选项不会显示被包含代码中的数据行。

程序8-4 启用SOURCE2 选项以显示源代码

```
%include "C:\temp\initdata.sas" /source2;
```

运行上面的代码后，在输出日志中可以看到如下内容：

```
NOTE:  %INCLUDE(水平 1)文件 C:\temp\initdata.sas 是文件
C:\temp\initdata.sas。
128 + data mydata;
129 + input x @@;
130 + infile datalines;
131 + datalines;
```

当我们需要包含多个文件时，可以使用空格或逗号将多个文件同时包含进来，比如：

```
%include "C:\temp\initdata.sas","C:\temp\printdata.sas";
```

如果操作系统支持聚合存储形式（如文件目录）的文件容器，我们也可以使用filename语句指向该目录，然后用括号将容器中的多个文件名以空格或逗号分隔的形式包含进来，这样就程序员就不必为每个文件提供全路径信息。比如下面的例子将C:\temp中的两个文件包含进来。

```
filename initdata "C:\temp";
%include initdata(initdata.sas printdata.sas);
```

注意：在Base SAS图形交互界面的命令输入框中可使用%INCLUDE命令或RECALL命令来包含SAS程序或者重新召回上次提交的源代码，其中命令输入框中可直接调用已经解析的宏函数，它们是图形交互界面的便捷操作功能。

8.2　程序中动态扩展代码

在SAS程序中，不但可以直接写代码，也可以将字符串文本当作源代码去动态编译执行。CALL EXECUTE例程的功能就是将其参数argument解析为SAS源代码，然后在本DATA步内或本DATA步运行结束后执行解析出来的源代码。其基本语法如下：

```
CALL EXECUTE(argument);
```

其中参数*argument*可以是一个字符串常量，或者是包含宏调用或SAS语句的文本表达式，也可以是一个字符型变量，或是其他可以被解析为宏文本表达式或SAS语句的字符型表达式。对于使用单引号和双引号括起的参数，在解析运行方面具有不同的行为：如果参数是由双引号括起来的，则参数会在当前DATA步编译之前就进行解析；如果参数是由单引号括起来的，则参数会在程序运行期间才解析。因此，参数中包含的宏代码在这两种情况下具有不同的执行逻辑（见程序8-5）。

程序8-5　CALL EXECUTE单引号字符参数

```
%let B=1;
data _null_;
  put "Before Invoke";
  tabname="sashelp.prdsale";
  v=symget("B");
  put v=;
  /*对宏变量进行赋值，会立即执行后执行下一步*/
  call execute( '%let B=2;' );
  v=symget("B");
  put v=;
  put "After Invoke";
run;
```

系统输出结果为

```
Before Invoke
v=1
v=2
After Invoke
```

但如果参数是使用双引号括起来的，即代码改为 call execute("%let B 2;"); 则系统输出将会不同，运行结果中两个 v 都等于 2，说明双引号括起来的参数确是在 DATA 步编译之前就被解析执行了，才导致 symget("B") 输出发生变化。

如果 CALL EXECUTE 子例程的参数中既包含 SAS 宏代码，也包含非宏的 SAS 代码时，SAS 编译器将会如何处理呢？参考如下代码（见程序 8-6）。

程序8-6 CALL EXECUTE 字符参数包括非宏SAS 代码

```
%let B=1;
data _null_;
  put "Before ZInvoke";
  tabname="sashelp.prdsale";
  v=symget("B");
  put v=;
  call execute('
    %let B=2;
    data _null_;
      put "Second DATA step";
      b=symget("B");
      put b=;
    run;
    ');
  v=symget("B");
  put v=;
  put "After Invoke";
run;
```

此时系统输出结果如下：

```
Before Invoke
v=1
v=2
Second DATA step
b=2
```

说明 %LET B 2; 是执行到 CALL EXECUTE 时才执行，而参数中解析出来的非宏 SAS 代码则被延迟到当前 DATA 步运行结束后才调用。但如果我们把 CALL EXECUTE 参数的单引号和双引号对调一下，结果会是什么呢？

```
call execute( "%let B=2; data _null_; put 'Second DATA step';
  b=symget('B'); put b=; run;" );
```

则系统输出 v =全部为 2，原因是参数中的 %LET B 2; 语句在进入主 DATA 步之前就被执行了，而参数解析出的 DATA 步 SAS 代码则被延迟当前步运行结束时才调用。这种行为说明不管 CALL EXECUTE 的参数是单引号还是双引号括起来的，其生成的非宏 SAS 代码都是被延迟到该 DATA 步运行结束之后执行的。在程序 8-7 中不管 execute 参数是单引号还是双引号括起来，PROC PRINT 步总是被延迟执行。

程序8-7 CALL EXECUTE 执行时序控制

```
%macro printds;
  proc print data=sashelp.class;
  run;
%mend;
data _null_;
  put "Before Invoke";
  call execute("%printds"); /* '%printds' 亦同*/
  put "After Invoke";
run;
```

上面的代码在逻辑上等价于程序 8-8 的执行时序：

程序8-8 等价的执行时序

```
data _null_;
  put "Before Invoke";
  put "After Invoke";
run;
proc print data=sashelp.class;
run;
```

除了常量字符串，CALL EXECUTE 也可接受字符型变量作为参数，如程序 8-9 所示。

程序8-9 字符变量作为CALL EXECUTE 参数

```
data _null_;
  put "Before Invoke";
  code='%printds;';
  call execute(code);    /*字符型变量作为参数*/
  put "After Invoke";
run;
```

更多的时候，我们会将字符型表达式作为参数，实现在 DATA 步中动态生成参数调用某个宏函数，实现根据某个数据集的内容动态调用宏函数的效果。下面笔者设计的几行代码（见程序 8-10）可将当前 SAS 系统逻辑库 SASHELP 中上百个数据集的描述信息打印出来，相当于将某个数据库中所有的表结构给打印出来了，技巧就是对输入数据集 SASHELP.VTABLE 的每一行记录调用一次 SAS 宏。

程序8-10 对数据表中的每一行数据执行某个 SAS宏实现处理控制

```
%macro printcnt(ds=sashelp.class);
  proc contents data=&ds;
    title &ds;
  run;
%mend;
data _null_;
  set sashelp.vtable;
  if memtype="DATA" AND libname="SASHELP" then do;
    call execute( '%printcnt(ds=' || trim(left(libname)) || '.' ||
      trim(left(memname)) || ');' );
  end;
run;
```

8.3 动态执行外部命令

动态执行外部命令包括全局 X 语句、CALL SYSTEM 例程和 SYSTEM 函数、%SYSEXEC

宏语句3种方法。

1.X全局语句

在SAS程序中，我们可以很方便地跟底层操作系统进行交互。其中X语句就是在SAS当前会话中执行外部操作系统命令的全局语句。在Linux版本的SAS中，终端用户G很方便地用X命令对SAS输出进行清屏，即X clear；比如，下面的x语句可以创建目录。

```
x 'md C:\mycodes';
```

执行X语句后会出现一个DOS控制台窗口并停留在屏幕上，如果希望执行完某个操作系统命令后关闭DOS控制台窗口，可使用DOS命令连接符 & 并执行EXIT命令自动退出DOS命令窗口。

```
x 'md C:\mycodes & exit';
```

利用X全局语句，我们可以将一个大型分析项目分解为若干子任务，然后在主程序中按需调用子任务来组织项目代码。比如，我们已经在C:\temp\initdata.sas中写好了准备数据的代码（见程序8-11），该代码生成mydata数据集到C:\temp目录中。

程序8-11　待执行的SAS 代码片段

```
libname mylib "C:\temp";
data mylib.mydata;
  input x @@;
  infile datalines;
  datalines;
1 3 5 7 9 2 4
6 8 10
run;
```

随后我们在主程序中需要使用程序生成数据并进行后续分析工作，则可以调用X语句运行该SAS程序（见程序8-12）。

程序8-12　在SAS 代码中调用另一个SAS 进程执行SAS 代码片段

```
x 'sas C:\temp\initdata.sas & exit';

libname mylib "C:\temp";/*此时 C:\temp 目录中已经有数据集了*/
proc print data=mylib.mydata;
  title;
run;
```

由于SAS中的X语句是全局语句，如果X语句出现在DATA步内时，它会在DATA步编译完成后自动执行。不要企图在DATA步内用IF语句实现条件执行X语句，要实现条件执行外部系统命令必须用CALL SYSTEM例程或SYTEM函数。

2.SYSTEM例程和SYSTEM函数

CALL SYSTEM与X全局语句的区别是它可以在数据步内被条件调用。系统函数SYSTEM与该CALL SYSTEM例程类似，但SYSTEM函数可以返回所执行的外部命令

的状态码。CALL SYSTEM 子例程和SYSTEM函数能执行的命令长度不得超过1024字符。

下面的几行 SAS 代码（见程序 8-13）能将操作系统上系统目录 C:\Windows 中的所有文件名收集到 SAS 数据集中供文件异常分析，其中如果存在 system.ini 文件的话，SAS 将弹出 Windows 记事本程序打开该配置文件。

程序8-13　在SAS代码执行前后，运行外部操作系统命令

```
x 'dir C:\windows /b > C:\temp\files.txt & exit'  ;

data files;
  infile "C:\temp\files.txt" end=last;
  input filename $32.;
  if filename="system.ini" then
    call system("notepad C:\windows\system.ini");
run;

x 'del C:\temp\files.txt /f & exit' ;
```

甚至可以编写简单的网络探测器，检测互联网中的特定机器是否在线（见程序 8-14）。对于不需要检测的主机只需要将 flag 标志设置为 0 即可。

程序8-14　简单的网络嗅探器

```
data conn;
  input flag $1. host $32.;
  if flag="y" then do;
    offline=system( "ping " || host || " & exit" );
  end;
  datalines;
y www.baidu.com
y www.sina.com
y www.sohu.com
n www.google.com
run;
proc print;run;
```

程序输出如图 8-2 所示。

Obs	flag	host	offline
1	y	www.baidu.com	0
2	y	www.sina.com	1
3	y	www.sohu.com	0
4	n	www.google.com	

图 8-2　本机网络连通性数据

3.%SYSEXEC宏语句

如果需要在SAS宏代码或开型代码中执行外部系统命令，可以使用系统宏%SYSEXEC语句来执行操作系统命令。该宏语句%SYSEXEC 执行命令返回的状态码可从系统宏变量&SYSRC 中读取（见程序 8-15）。

程序8-15 在SAS 宏中执行外部系统命令

```
%SYSEXEC DIR C:\WINDOWS /B > C:\TEMP\FILES.TXT & EXIT;
%PUT &SYSRC; /*执行成功返回 0*/

%SYSEXEC NOTEPAD C:\TEMP\FILES.TXT;
%PUT &SYSRC; /*执行成功返回 0*/

%SYSEXEC ; /*弹出 DOS 窗口*/
```

其中最后一个无参数调用会弹出 DOS 命令窗口等待用户输入，直到用户输入 EXIT 命令返回 SAS 会话。

上面 3 种执行系统命令的行为方式，受 2 个 SAS 系统选项的控制和影响，分别用于控制 DOS 窗口的行为和命令执行时序。

（1）XWAIT/NOXWAIT：用于控制 SAS 执行外部命令的 DOS 命令窗口是否需要人为输入 EXIT 命令才关闭；默认是选项是 XWAIT；NOXWAIT 意味着 DOS 命令窗口在命令执行后自动关闭。

（2）XSYNC/NOXSYNC：用于控制 SAS 是否等待执行命令启动的外部应用执行完毕才返回 SAS 会话环境。这两个选项分别控制同步和异步模式：同步模式意味着 SAS 会话会等待所执行的命令执行完毕后才返回 SAS，为默认选项；异步模式则表示 SAS 不必等待外部应用执行完毕，可直接返回。同步模式一般用来需要等待外部命令执行完毕生成文件或数据才能继续的场景。

上面 2 个选项可以组合出 4 种情形，但前者是控制 DOS 命令窗口的行为，后者是控制 SAS 是否需要等待应用执行完毕的时序控制。比如，在 SAS 代码中静默执行一个外部命令无须等待返回，可设置为

```
options noxsync noxwait;
x 'md c:\temp\foo';
```

由于 SAS 可以运行在多种 Windows 或 Linux/Unix 平台上，而不同平台上的文件路径表示方式不同，此时在 SAS 代码中可通过检测系统宏变量 &SYSSCP 和 &SYSSCPL 来标识不同的平台类型，从而编写移植性较好的 SAS 代码。比如，在 Windows 7 Professional 64 位系统上 &SYSSCP &SYSSCPL 的输出值为 WIN 和 X64 7PR，而在 Windows 10 Professional 64 位平台上的输出值为 WIN X64_10PRO。

总的来说，SAS 代码的组织技巧包括静态文件包含，程序中动态扩展代码以及动态执行外部 SAS 程序等几种方式。SAS 代码不必像 Java 类文件那样呆板，用户可随意组织 SAS 代码源文件。

第9章

文件读写

SAS 作为面向分析的第四代计算机语言，它提供了强大的文件读写能力。它不但支持通用编程语言 C/C++ 那样的低级文件读写能力，它还提供面向数据的预定义格式化读写能力。本章主要讲述 SAS 语言中和传统编程语言 C/C++ 和 Java 类似的文件读写特性以及部分 SAS 特有的高级读写特性。

9.1　二进制文件读写

计算机文件本质上都是磁盘系统上的二进制字节流，因此读写二进制文件是掌握文件读写的关键。SAS 读写二进制文件分别使用 infile 和 file 语句，配合语句选项 recfm=n 进行；其中语句选项 recfm 有多个可能的值，其中 n 表示按照没有记录边界的字节流方式读写文件；此时如果没有指定逻辑记录长度 LRECL，其默认值为 32767 字节。当文件打开后使用 input/put 语句执行真正读 / 写数据时，还需配合特殊格式 ib1. 进行。需要注意一点是，SAS 的二进制读写方式对命名管道和管道设备类型无效。

程序 9-1 创建二进制文件 C:\abc.dat，并写入十进制值为 77 的一个字节，该数值的十六进制为 4D，即英文字母‘M’的 ASC Ⅱ码点。在给变量 byte 赋值时，我们也可以使用十六进制方式赋值，如 byte=4Dx;

程序9-1　往二进制文件中写入一个字节

```
data _null_;
  file "C:\abc.dat" recfm=n;
  byte=77; /*或 byte=4Dx*/
  put byte ib1.;
run;
```

程序 9-2 实现往二进制文件 C:\abc.dat 中写入整个 ASC Ⅱ 码表，其中 ASC Ⅱ 码表的数值介于十进制的 0~127（十六进制的 00-7F），其中 32-126 为可显示字符，其余为控制字符。

程序9-2　往二进制文件中写入整个ASC II码表

```
data _null_;
  file "C:\abc.dat" recfm=n;
  do i=0 to 127;/*ASC II 32-126 为可显示字符*/
    byte=i;
    put byte ib1.;
  end;
run;
```

我们也可以使用特殊的内嵌数据区来读取数据，程序 9-3 从源代码内嵌数据行中读入十六进制文本 415A617A3039207E 并将它们转换成字节写入二进制文件 C:\abc.dat。注意 input 语句读取数据时使用了十六进制方式连续读取，而“if byte^=.;”一行代码是用来防止 SAS 将输入缓冲区的空白字符写入到目标文件。

程序 9-3　将数据行中的十六进制数据写入二进制文件

```
data _null_;
  infile datalines;
  input byte hex2. @@;
  if byte^=.;
  /*内嵌数据行默认增长 40 记录，读取外部文件则不需要此控制*/

  file "C:\abc.dat" recfm=n;
  /*打开文件并将读入的字节写入二进制文件 C:\abc.dat*/
  put byte ib1.;

datalines;
415A617A3039207E
run;
```

我们可用如下程序逐个字节读取刚才写到磁盘上的二进制文件 C:\abc.dat 并将其用十进制表示对应的字符，以及十六进制表示写入数据集 work.mydata 中（见程序 9-4）。

程序 9-4　从二进制文件中按字节读取数据并构建数据集

```
data mydata;
  infile "C:\abc.dat" recfm=n;
  input byte ib1. @@;

  put byte hex2.;

  char=byte(byte);/*将数据转成字符输出到 SAS 数据集*/
  hex=put(byte, hex2.);/*输出数据的十六进制表示*/
run;
proc print data=mydata; run;
```

运行该代码后系统输出如图 9-1 所示。

Obs	byte	char	hex
1	65	A	41
2	90	Z	5A
3	97	a	61
4	122	z	7A
5	48	0	30
6	57	9	39
7	32		20
8	126	~	7E

图 9-1　从二进制文件读入数据

二进制文件读取也可以指定每次读取的字节数，比如中国券商软件通达信的日线数据就是以二进制格式存储的，我们可以通过 input 语句指定 ib4. 或 ib8. 格式来一次读取 4

或者 8 个字节，用户可相应配合输出格式 hex8. 或 hex16. 进行调试和输出。掌握了二进制文件读写可以非常方便地解析计算机上任何格式的数据，而且用 SAS 语言编写程序来读写数据代码非常简洁高效，与 C/C++ 等通用语言比起来更加出众。笔者使用 SAS 程序读取通达信交易软件的日线数据只有区区 15 行，就能把二进制的交易数据迅速转换成 SAS 数据集供进一步分析，如果采用其他语言编写同等程序的话其代码量将远远超过 SAS 代码。

9.2　文本文件读写

SAS 在 Windows 平台上的默认记录格式（Record Format）为 recfm=V，表示按照变长记录格式读写，该选项的其他格式还包括 F 和 P，分别表示定长记录格式和打印格式。在 SAS 中读写文本文件代码极其简单，程序 9-5 往文本文件 C:\abc.dat 中写入两行文本。

程序9-5　将字符数据写入文本文件

```
data _null_;
  file "C:\abc.dat";
  put "The Power to Know";
  put "The Only Constant is Change";
run;
```

由于文本文件通常涉及字符编码这一概念，字符编码的本质就是将字符语义跟它对应的编码值（即码点）进行映射，而包括特定字符的编码集合称为字符集，根据表示一个字符所使用的字节的多少，编码字符集可分为单字节字符集（SBCS）、双字节字符集（DBCS）和多字节字符集（MBCS）三大类。现在普遍使用的 Unicode 编码系统试图统一所有的字符编码方式，并在具体编码实现上引入变长字符编码方式 UTF-8、UTF-16 以及 UTF-32 等。

在计算机发展的早期并没有考虑到各国文字编码问题，由美国国家标准协会（ANSI）制定的美国信息标准交换码（ASC II）用 7 位的二进制数（0x00-0x7F）表示大小写的英文、数字和基本标点符号以及控制字符，其最高位被用作奇偶校验位。后来随着计算机的发展各国纷纷基于 ANSI 标准扩展了自己的 ASC II 编码，其秘密就是利用码点在 0x80-0xFF 范围内的两个字节表示自己国家或地区的字符编码，从而衍生出中国大陆使用的 GB2312 编码、中国台湾地区使用的 BIG5 编码以及日本使用的 Shift-JIS 编码等系统，但这些扩展 ASC II 编码互不兼容。

对于中国大陆使用的国家标准汉字编码，主要是基于 ANSI 编码的中文扩展 ASC II 编码，即收录 6763 个汉字和 682 个拉丁、希腊以及日俄字母的 GB 2312 编码。它的前 127 个字符与 ASC II 编码兼容，而大于 127 的两个字符连在一起用于表示单个汉字或符号：第一个字节为 0xA1-0xF7；第二个字节为 0xA1-0xAE。然而由于汉字太多，于是后来对第二个字节也不再要求大于 127，这样 1995 年 12 月 15 日就扩展为收录 21886 个汉字和图形符号的汉字内码扩展规范 GBK 编码（其中 21003 个汉字、部首或构件，883 个

图形符号）。2000年3月17日和2005年11月8日发布的GB 18030-2000和GB 18030-2005国标字符集分别扩容至27533和70244个汉字，它们分别兼容Unicode中日韩统一表意文字扩展A区和B区，并包括各个少数民族的文字。中文编码GB2312、GBK、GB18030是逐渐扩展、后向兼容的官方中文编码系统。目前，由于全球UNICODE编码标准的推广，各国扩展ASC II编码都有逐渐向UNICODE编码统一的趋势。

作为国际化软件系统，SAS支持几乎所有平台的主流编码，涵盖各种UNIX系统和Windows系统。其中Latin1/Latin9分别表示ISO的西欧/欧洲标准，主要用于UNIX/LINUX平台；wlatin1/wlatin2表示Windows的西欧/中欧标准，用于Windows平台。表9-1列出了SAS常用语言和平台的数据编码方式。

表9-1 SAS常用语言和平台的数据编码方式

英文/法文	简体中文	繁体中文	日文	韩文
waltin1/wlatin2	gb2312/gbk	big5	shift-jis	
	ms-936	ms-950	ms-932	ms-949
latin1/latin9	euc-cn	euc-tw	euc-jp	euc-kr
	ibm-935	ibm-937	ibm-939	ibm-933
open_ed-924	open_ed-935	open_ed-937	open_ed-939	open_ed-933
Unicode编码系统：utf-8 utf-16be utf-16le utf-32be utf-32le				

在SAS代码中读写文件时如果不指定编码方式，默认使用操作系统的ANSI编码方式，对于中文就是GB 2312和GBK兼容编码。随着UNICODE的兴起，大家也常使用utf-8或utf-16le等平台默认的Unicode编码方式进行文本文件读写。下面的例子使用UTF-8编码方式输出包含中文的两行文本到C:\abc.dat中。

程序9-6 将字符数据按照指定编码写入文本文件

```
data _null_;
  file "C:\abc.dat" encoding="utf-8";
  put "慧识力量";
  put "The Only Constant is Change";
run;
```

对于UTF-8编码方式，SAS系统会自动输出字节顺序标志BOM到文件的头部，也就是说它输出文件的前3个字节是标志字节EF BB BF，用于明确标记该文件的编码为UTF-8，供别的系统或应用在读取时能够识别它。程序9-7以文本方式打开前面生成的文本文件，按行读取数据并打印在SAS日志窗口中。

程序9-7 按行读取外部文本文件到缓冲区和数据集

```
data mydata;
  length line $ 255;
  infile "C:\abc.dat" delimiter='0D0A'x ;/*以回车换行符分隔*/
  input line;
  put line;
run;
```

系统输出如下：

```
慧识力量
The Only Constant is Change
```

需要注意的一点是，由于SAS为了支持各种常见的文本格式数据读取，其默认读取方式为变长记录格式（recfm=V），它默认使用空格和TAB字符作为记录之间的分割符，因此如果要像C/C++那样按行读取文本，需要在infile语句上显式指定使用回车换行作为记录分隔符（delimiter='0D0A'x）并设置读入文本行的最大长度为足够长，如255。

对于一个外部文件，如果根本不知道每一行文本有多长，该如何读取这些变长的文本行呢？SAS提供了特殊的读取格式$varying，并指定其最大支持的文本长度即可；同时对于每一行读入的值，可通过length=指定状态变量linelength来获取读入的实际长度（见程序9-8）。

程序9-8 读取每行不定长的外部文本文件

```
data mydata;
  infile "C:\temp\abc.dat" length=linelength;
  input line $varying32767. linelength;
  put line= linelength= ;
run;
```

系统输出如下：

```
line=慧识力量 len=8
line=The Only Constant is Change len=27
```

在file/infile语句上，我们也可以指定读写外部文件的最大逻辑记录长度LRECL。这个参数的有效值为1~32767的任意整数。它可以用十进制或十六进制数表示，或者指定MIN或MAX（其中MIN=1、MAX=32767）；使用十进制表示时也可同时指定存储单位K、M、G或T，分别表示存储单位Kb、Mb、Gb或Tb。

LRECL是按字符计算的，但运行时系统会转换为实际读入文件编码有关的真实字节数。比如，指定27个字符，如果外部文件abc.dat是ANSI格式，运行时日志里显示LRECL=27；但如果abc.dat是UTF-8格式的，一个汉字最多可能需要4个字节存储，所以在运行时日志中显示为LRECL=108，是用户指定的LRECL值的四倍。infile语句的LRECL选项需谨慎指定，如果LRECL太小可能导致数据截断错误，比如程序9-9由于指定为LRECL=25，其输出的第2行文本会被截断2个字节。

程序9-9 指定读入文件的LRECL 长度

```
data mydata;
  infile "C:\abc.dat" length=linelength lrecl=25; /*要求 lrecl>=27*/
  input line $varying32767. linelength;
  put line= linelength= ;
run;
```

如果我们需要自己控制数据解析，则不能给input语句赋予任何参数，然后可使用系统内部变量_infile_来访问读入缓冲区的数据。下面的代码（见程序9-10）生成一个数据集，每一行记录是从缓冲区_infile_文本中解析出的一个单词。这一机制为我们从文本文件中读取格式灵活的数据成为可能，结果如图9-2所示。

程序9-10　自定义缓冲区数据解析处理

```
data mydata;
  infile "C:\abc.dat" delimiter='0D0A'x ;
  input;

  length word $ 32;
  i=1;
  word= scan(_infile_, i, ' ');
  do while(word ^=' ');
    put word;
    output;
    i=i+1;
    word= scan(_infile_, i, ' ');
  end;
  drop i;
run;
proc print data=mydata;run;
```

Obs	word
1	慧识力量
2	The
3	Only
4	Constant
5	is
6	Change

图 9-2　自定义数据解析过程

9.3　顺序读取多个文件

SAS 语言支持同时从多个文件中读取数据到同一数据集中，其精妙之处就是利用 filename 语句创建单个文件引用，但实际上它包含了多个物理文件。程序 9-11 顺序读取两个文件内容并生成单个数据集。

程序9-11　顺序读取多个文件示例

```
/*生成文件一 abc.dat*/
data _null_;
  file "C:\abc.dat" ;
  put "The Power to Know";
  put "The Only Constant is Change";
run;
/*生成文件二 def.dat*/
data _null_;
  file "C:\def.dat" encoding="utf-8";
  put "慧识力量";
  put "唯有变化，才是永恒";
run;

/*读入多个文件的数据到同一目标数据集*/
filename files ( 'C:\abc.dat','C:\def.dat') encoding="utf-8";
data mydata;
```

```
  length line $ 255;
  infile files delimiter='0D0A'x ; /*files实际上指向多个文件*/

  input line;
  put line=;
run;
```

结果数据集 mydata 包含 4 行文本，如图 9-3 所示。

Obs	line
1	The Power to Know
2	The Only Constant is Change
3	慧识力量
4	唯有变化，才是永恒

图 9-3　顺序读取多个文件

另外，SAS 的 filename 语句还支持文件通配符，这样我们就可以从磁盘上某个目录下批量读入多个文件了。比如：filename files ('C:*.dat') encoding="utf-8" 可以读取 C 盘根目录下的所有后缀为 DAT 的文件。

使用 filename 语句来读取多个外部文件需要我们预先知道文件名，但如果文件列表来自某个数据集或者外部文件，我们又该如何进行批量文件读取呢？

比如，我们希望在单个数据步内顺序读取多个文件，可使用 infile 语句并指定一个 filevar 变量来实现，而 filevar 所指向的变量的值是从某个文件列表或人为指定的。为了判断每个数据文件读取是否结束，我们使用 end= 选项来输出结束标志到某个状态变量 eof 中，然后在该 infile 语句后就可以使用 eof 变量来判断是否读到了单个文件的最后。程序 9-12 完整展示了这种多文件读取的灵活机制。

程序9-12　从文件清单中指定的多个文件中读取数据

```
/*生成 list.dat 文件*/
data _null_;
  file "C:\list.dat" ;

  put "C:\abc.dat";
  put "C:\def.dat";
run;

data mydata;
  infile "C:\list.dat";
  length fileloc $ 256;
  input fileloc $; /*从 list.dat 中读入文件路径到 fileloc 变量中*/

  /*将 fileloc 变量传递给读取数据的 infile 语句，并指定分隔符和结束标志变量*/
  infile myfile filevar=fileloc delimiter='0D0A'x end=eof ;
  put " Begin-- " fileloc ;
  do while(not eof);
    length line $255;
    input line;
    put line ;
```

```
    output;
  end;
  put "--End-- " fileloc ;
run;
```

系统输出如下：

```
--Begin-- C:\abc.dat
line=The Power to Know
line=The Only Constant is Change
--End-- C:\abc.dat

--Begin-- C:\def.dat
line=慧识力量
line=唯有变化，才是永恒
--End-- C:\def.dat
```

如果文件列表已经在某个SAS数据集某列内存在，如假定该列名为fileloc，则我们也可以非常简单地利用SET语句从该数据集中获取filevar值（见程序9-13）。

程序9-13 从数据集变量中指定的多个文件中读取数据

```
data filelist;
  infile datalines;

  length fileloc $ 300;
  input fileloc;
  datalines4;
C:\abc.dat
C:\edf.dat
;;;;
run;

data mydata;
  set filelist;

  /*将 fileloc 变量传递给读取数据的 infile 语句，并指定分隔符和结束标志变量*/
  infile myfile filevar=fileloc delimiter='0D0A'x end=eof ;
  put "-Begin-- " fileloc ;
  do while(not eof);
    length line $255;
    input line;
    put line=;
    output;
  end;
  put "--End-- " fileloc ;
run;
```

即使在读取多个文件时，如果我们需要完全控制对缓冲区数据的解析，依然可以像前面一样使用内部变量_infile_进行自定义解析。下面的程序（见程序9-14）演示了从两个外部文件读取时，使用SCAN函数对每一行数据进行自定义解析的过程。

程序9-14 读取多个文件时自定义缓冲区解析逻辑

```
data _null_;
  do i = 1 to 2;
    /*生成列表文件 list.dat，包含两行数据指向 dat01.dat, dat02.dat */
    file 'C:\temp\list.dat';
    fname= 'C:\temp\dat' || put(i,z2.) || '.dat';
```

```
    put fname;

    /*生成数据文件 dat01.dat dat02.dat 数据，每一行包含 i j 两列*/
    file datfiles filevar=fname;
    do j = 1 to 3;
      put i j;
    end;
  end;
run;

data mydata;
  /*读取列表文件 list.dat 并将缓冲区变量设置为 fname*/
  infile 'C:\temp\list.dat' _infile_=fname;
  input;

  /*对于每一个文件 fname 读入并解析它的缓冲区变量 _infile_*/
  infile datfiles filevar=fname end=eof;
  put "--begin-- " fname;
  do while(^eof);
    input;
    put _infile_;/*输出到 SAS 日志*/

    /*自己解析缓冲区数据输出到数据集中*/
    i=scan(_infile_, 1," ");
    j=scan(_infile_, 2," ");
    output;
  end;
  put "--end-- " fname;
run;

proc print data=mydata; run;
```

系统生成数据集 mydata 如下（见图 9-4），其中前 3 行来自第一个文件；后 3 行来自第二个文件。

Obs	i	j
1	1	1
2	1	2
3	1	3
4	2	1
5	2	2
6	2	3

图 9-4　从多个文件读取数据并解析

9.4　并行读取多个文件

SAS 也可以在单个数据步内同时打开多个文件，然后从每个文件中并行读入数据，

直到最短的那个文件被读完为止。利用这一特性，我们可以从多个文件中读取数据到数据集同一观测的不同列上，如下程序（见程序9-15）生成的数据集中，English和Chinese两列数据分别来自不同的数据文件abc.dat和edf.dat（见图9-5）。

程序9-15　并行读取多个文件中的数据建立观测

```
data mydata;
  infile 'C:\temp\abc.dat' delimiter='0D0A'x   ;

  length english $ 255;
  input english;

  infile 'C:\temp\def.dat' delimiter='0D0A'x encoding="utf-8";

  length chinese $ 255;
  input chinese;
run;
proc print data=mydata;run;
```

Obs	english	chinese
1	The Power to Know	意识力量
2	The Only Constant is Change	唯有变化，才是永恒

图9-5　并行读取多个文件

9.5　共享缓冲区读写

SAS对某个文件的读写可发生在同一个数据步内，infile语句会有输入缓冲区，而file会有输出缓冲区。如果我们希望对某个文件进行修改，则我们可以直接让infile打开文件时设置为共享缓冲区模式，这样file的put语句就可重用该输入缓冲区，将数据直接写到磁盘文件上。如下程序（见程序9-16）演示如何读入外部文件abc.dat数据并将数据进行反转输出，结果如图9-6所示。由于反转后的数据写回输入文件中，该程序每执行一次数据就发生一次反转。

程序9-16　共享同一缓冲区读写示例

```
filename myfile "C:\temp\abc.dat";
data mydata;
  length line $ 255;
  infile myfile delimiter='0D0A'x sharebuffers;
  file myfile; /*让 infile 和 file 共享缓冲区*/

  input line;
  line reverse(line);
  put line ; /* put 语句直接将输入缓冲区(而非输出缓冲区)数据写到磁盘*/
run;
proc print data mydata ;run;
```

Obs	line
1	wonK ot rewoP ehT
2	egnahC si tnatsnoC ylnO ehT

图 9-6 读写共享缓冲区

本章讲述了在 SAS 中如何读写二进制文件和文本文件，读出的数据可灵活构造 SAS 数据集以供分析。对于文本文件，SAS 在读写时可指定文件编码并自定义数据解析规则。SAS 还支持多文件顺序处理和并行处理，最后简单演示了如何利用共享缓冲区修改外部文件。

第10章

按位运算

现代计算机系统采用的二进制系统可以追溯到1679年受到中国《易经》启发的德国数学家莱布尼茨，他在1703年发表的论文《论只使用符号0和1的二进制算术，兼论其用途及它赋予伏羲所使用的古老图形的意义》中提出了二进制。现代计算机在硬件底层基本上都采用二进制系统进行数据存储和运算，组成二进制数的每一位为0或1，称为一个位或比特（Bit），由数字电路中的逻辑门直接实现。图10-1是由8个逻辑"位"组成的存储单元"字节"示意图，相当于由8盏灯连在一起组成的一个复合信号单元，每盏灯都可以是开/关（1/0）两种状态，则8盏灯串在一起可表示256种不同信号。图10-1中的二进制01010110表示十进制数为86，表示十六进制数为56。如果它表示一个ASC II字符，则是ASC II码表中的大写字母'V'。



图10-1　字节与比特

现代计算机存储器（如内存、硬盘和各种存储卡）的最小存储单元是字节，而编程语言的基本数据类型系统通常基于字节而构建，包括字符、整数和浮点数等系统表达方式。为文本数据构建的ASC II/EBCDIC以及后来的Unicode/UTF-8等编码系统应运而生，为数值数据构建的数据类型系统，包括整型系统（短整型-16位、整型-32位、长整型-64位）和浮点型（单精度-32位、双精度-64位），通常由IEEE 754规范定义的二进制浮点类型格式及其变体来定义。

位操作是计算机系统上的最小粒度的操作处理，实际上所有的计算机处理最终都是基于比特位之上构建的各种基础运算来完成。现代计算机架构中"位运算"通常与加法运算的速度相同，它比乘法运算要快。一些早期的计算机通用语言(如C/C++)强大的原因之一是它能从最小的存储粒度"位"上操纵数据，其核心就是位操作和指针处理，那些计算机通用语言因此可控制各种硬件设备。

10.1　按位运算

本章重点探讨SAS中如何实现位操作，以及使用位操作完成，如数据加密和基于二

进制的复杂算法设计，位运算在图像处理和布尔密集型计算中也有广泛的使用。位运算本身规则简单，基本位操作运算有如下几种方式。

（1）与操作（AND）：对两个等长的二进制数，相应的二进制位都为 1，则结果中相应二进制位为 1。如 0101 0110 AND 0000 1111=0000 0110。

（2）或操作（OR）：对两个等长的二进制数，相应的二进制位只要有一个为 1，则结果中相应二进制位为 1。如 0101 0110 OR 0000 1111=0101 1111。

（3）异或操作（XOR）：对两个等长的二进制数，相应的二进制位如果不同，则结果位为 1。如 0101 0110 XOR 0000 1111=0101 1001。

（4）取反操作（NOT）：对单个二进制数的每一个二进制位执行取反操作，即 0 变成 1，1 变成 0。如 NOT 0101 0110 = 1010 1001。

（5）移位操作：包括左移位 LSHIFT 和右移位 RSHIFT。将二进制数所有比特位向左或右移动指定位数，将溢出的部分舍去，空缺的部分填入 0 值。

如左移 3 位：0101 0110 << 3 = 1011 0000，右移 3 位：0101 0110 >> 3= 0000 1010。

需要注意的一点是，这里的按位逻辑操作与布尔代数中的逻辑操作不同，它是二进制数中的相应比特位之间进行逻辑 AND/OR/XOR/NOT 操作，而非整体之间进行布尔逻辑操作，故称按位运算（Bitwise Operation）。

10.2 实现方法

在 SAS 中有多种方法实现按位运算，这里主要讲解其中的两种。

（1）利用 INPUT/PUT 函数配合输入输出格式 $BINARY. 进行位操作。在 SAS 变量和外部的字符表达形式之间，可用输入输出格式进行转换。RANK/BYTE 系统函数也可在字符码点值和字符变量之间进行转换，参考如下代码（见程序 10-1）。

程序10-1 输入输出格式配合INPUT/PUT 函数实现数据表示和存储之间的转换

```
data _null_;
  a='V';

  b=put(a, $BINARY8.);    /*返回二进制表示*/
  c=input(b, $BINARY8.); /*二进制表示转成字符变量*/

  h=put(a, $HEX2.);       /*返回十六进制表示*/
  i=input(h, $HEX2.);     /*二进制表示转成字符变量*/

  o=RANK(a);              /*返回数字，即十进制表示*/
  e=BYTE(o);              /*数字转换为字符*/
  put a= b= c= o= e= h= i=;
run;
```

系统输出为 a=V b=01010110 c=V o=86 e=V h=56 i=V。

本方法主要利用格式化和隐性数据类型转换模拟二进制操作，它不是在变量字节存储层面上进行位操作，而是转成对应的字符表达后进行操作，最后改变变量存储本身。

比如字符 ‘A’ 和 ‘B’ 要进行 AND 操作，首先各自转成二进制表示的字符串“01000001”和“01000010”；其次利用加法中的自动类型转换（它们都隐性变为十进制数 1000001 和 1000010），加法运算结果为十进制数 2000011，利用 z8. 格式化输出后的字符串表达为“02000011”；再次利用字符串替换函数 translate 将 1 替换为 0，2 替换为 1 得到字符串“01000000”；最后利用二进制输入格式 $BINARY. 将它转成二进制数 01000000（即十六进制的 0x40），对应的字符表示为 ‘@’。程序 10-2 完整展示基于这种机制实现的各种位操作运算。

程序 10-2 格式化方法实现二进制按位操作

```
/*位操作方法1*/
data _null_;
  a='A';
  a2=put(a, $BINARY.);
  put a a2 '<- A';

  b='B';
  b2=put(b, $BINARY.);
  put b b2 '<- B';

  /* AND 操作*/
  s = translate( put(put(a, $BINARY.)+put(b, $BINARY.),z8.),'01','12');
  and_ab=input(s, $BINARY.);
  and_ab2=put(and_ab, $BINARY.);
  put and_ab  and_ab2 '<- A AND B';

  /* OR 操作*/
  s = translate( put(put(a, $BINARY.)+put(b, $BINARY.),z8.),'1','2');
  or_ab=input(s, $BINARY.);
  or_ab2=put(or_ab, $BINARY.);
  put or_ab  or_ab2 '<- A OR B';

  /* XOR 操作*/
  s=translate( put(put(a, $BINARY.)+put(b, $BINARY.),z8.),'0','2');
  xor_ab=input(s, $BINARY.);
  xor_ab2=put(xor_ab, $BINARY.);
  put xor_ab  xor_ab2 '<- A XOR B';

  /* NOT 操作*/
  s=translate( put(a, $BINARY.),'01','10');
  not_a=input(s, $BINARY.);
  not_a2=put(not_a, $BINARY.);
  put  not_a ' ' not_a2 '<- NOT A';

  /*左移位操作<< 2*/
  n=2;
  if n< length(trim(put(a, $BINARY.))) then
    s=substr(put(a, $BINARY.), n+1);
  else
    s=substr(put(a, $BINARY.), length(trim(put(a, $BINARY.))));
  ls_a=input(s , $BINARY.);
  ls_a2=put(ls_a, $BINARY.);
  put  ls_a   ls_a2 '<- A << ' n;

  /*右移位操作>> 2*/
  n=2;
```

```
  if n< length(trim(put(a, $BINARY.))) then
    s= trim(repeat("0",n 1)) || substr( put(a, $BINARY.), 1,
        length(trim(put(a, $BINARY.)))-n);
  else
    s="0";
  rs a=input( s, $BINARY.);
  rs a2=put(rs a, $BINARY.);
  put  rs a   rs a2 '<- A >> ' n;
  put;
  put a= hex. b= hex. and ab=hex. or ab= hex. xor ab= hex. not a=
    hex. ls_a= hex. rs_a= hex.;
run;
```

系统输出如下：

```
A 01000001 <- A
B 01000010 <- B
@ 01000000 <- A AND B
C 01000011 <- A OR B
_ 00000011 <- A XOR B
? 10111110 <- NOT A
_ 00000100 <- A << 2
_ 00010000 <- A >> 2

a=41 b=42 and_ab=40 or_ab=43 xor_ab=03 not_a=BE ls_a=04 rs_a=10
```

（2）利用SAS系统提供的6个函数bAND/bOR/bXOR/bNOT和bLSHIFT/bRSHIFT实现各种位操作。位操作函数的参数为介于0~2^{32-1}的整数，即0~4294967295的无符号整数，其中移位操作的位数参数则介于0~31，也就是说SAS位操作函数默认最大为4字节无符号整数，但我们可以通过分组实现对任意长度的字节序列进行位操作。程序10-3展示了双字节十六进制数ACE1以及右移5位后的结果0567。

程序10-3 基于SAS 按位操作函数实现位操作

```
data _null_;
  a=0ACE1x;
  put a  BINARY16. " " a HEX.;

  b=bRSHIFT( a, 5);
  put b BINARY16. " " b HEX.;
run;
```

系统输出：

```
10101100 11100001 0000ACE1
00000101 01100111 00000567
```

程序10-4完整展示了SAS位操作方面的全部功能，可查看输出来理解SAS中的位操作运算方法。

程序10-4 基于系统函数的二进制按位操作

```
data _null_;
  a='A';
  a2=put(a, BINARY.);
  put a a2 '<- A';

  b='B';
  b2=put(b, BINARY.);
```

```
put b b2 '<- B';

/* AND 操作*/
and_ab=bAnd(RANK(a), RANK(b));
and_ab2=put(and_ab, BINARY.);
put and_ab and_ab2 '<- A AND B';

/* OR 操作*/
or_ab=bOR(RANK(a), RANK(b));
or_ab2=put(or_ab, BINARY.);
put or_ab  or_ab2 '<- A OR B';

/* XOR 操作*/
xor_ab=bXOR(RANK(a), RANK(b));
xor_ab2=put(xor_ab, BINARY.);
put xor_ab  xor_ab2 '<- A XOR B';

/* NOT 操作*/
not_a=bNOT(RANK(a));
not_a2=put(not_a, BINARY.);
put  not_a ' ' not_a2 '<- NOT A';

/*左移位操作<< 2*/
n=2;
ls_a=bLShift( RANK(a), 2);
ls_a2=put(ls_a, BINARY.);
put  ls_a ls_a2 '<- A << ' n;

/*右移位操作>> 2*/
n=2;
rs_a=bRSHIFT( RANK(a), 2);
rs_a2=put(rs_a, BINARY.);
put  rs_a  rs_a2 '<- A >> ' n;

put;
put a= hex. b= hex. and_ab=hex. or_ab= hex. xor_ab= hex. not_a=
  hex. ls_a= hex. rs_a= hex.;
run;
```

系统输出如下：

```
A 01000001 <- A
B 01000010 <- B
64 01000000 <- A AND B
67 01000011 <- A OR B
3 00000011 <- A XOR B
4294967230  10111110 <- NOT A
260 00000100 <- A << 2
16 00010000 <- A >> 2

a=41 b=42 and_ab=00000040 or_ab=00000043 xor_ab=00000003
not_a=FFFFFFBE ls_a=00000104 rs_a=00000010
```

从输出结果可看出，二进制输出序列与前面的例子完全相同，但由于位操作函数默认输出为 4 个字节，因此其十六进制输出长度不同。可通过设置输出格式 HEX2. 来获得同样结果。

10.3 按位运算应用

1.位操作应用——加密处理

位操作的主要应用之一是在数据的二进制存储层面对数据进行变换，打破字节和字符边界对数据进行高级的加密功能。基于字母表移位变换的凯撒密码采用字母表偏移实现，在SAS中用几行简单的代码即可做到（见程序10-5），如HELLO WORLD移位为3的凯撒密码为KHOOR ZRUOG。

程序10-5 基于字符移位的恺撒密码

```
data _null_;
  text="HELLO WORLD";
  pass=translate(text, 'DEFGHIJKLMNOPQRSTUVWXYZABC',
    'ABCDEFGHIJKLMNOPQRSTUVWXYZ');
  put text "-> " pass;
run;
```

为演示目的，程序10-6利用密码“123456”对明文字符串“Hello World”在字节层面进行简单的逐段按位异或XOR形成密文。这种方法可用于简单的信息加密，实际项目中对信用卡号码，用户密码以及其他敏感数据的加密往往根据需要采用各种强加密算法在字节层面上对数据进行加密存储。基于XOR运算的特性，再次XOR操作即可从密文中解析出加密前的明文。

程序10-6 按位操作XOR 的简单加密

```
data _null_;
  a='Hello World';
  b='123456';
  put "明文: " a;
  put "密码: " b;
  length c $ 13;
  do i=1 to length(a);
    and_ab=bXOR(
      RANK(substr(a,i,1)),
      RANK(substr(b,mod(i,length(trim(b)))+1,1)));
    substr(c,i,1)=BYTE(and_ab);
    and_ab2=put(and_ab, $BINARY.);
  end;
  put "密文: " c ;
  length d $ 13;
  do i=1 to length(a);
    and_ab=bXOR(
      RANK(substr(c,i,1)),
      RANK(substr(b,mod(i,length(trim(b)))+1,1)));
    substr(d,i,1)=BYTE(and_ab);
    and_ab2=put(and_ab, $BINARY.);
  end;
  put "解密: " d ;
run;
```

系统输出为

```
明文：Hello World
密码：123456
密文：zVXYY_e\FYR
解密：Hello World
```

2.位操作应用——颜色分量

位运算在图形图像处理中也比较常见，由于计算机系统的色彩体系通常采用 4 字节的无符号整数表示，而图像处理需要对色彩和透明度进行分解处理。对于一个带有透明通道 Alpha 值的 RGB 颜色，我们可以利用位操作获取 Alpha 值和 RGB 3 种颜色分量，变换后也可以重新合成新的色值。程序 10-7 利用位操作从 4 字节表示的颜色值中获取 Alpha 值和 RGB 三原色分量。

程序10-7　对十六进制表示的颜色值进行分量处理

```
data _null_;
  color=0FF998877x;/*有透明通道Alpha值的RGB颜色表示*/

  alpha=bAnd(brshift(bAnd(color ,0FF000000x),24), 0FFx);
  red=bAnd(brshift(bAnd(color ,000FF0000x),16), 0FFx);
  green=bAnd(brshift(bAnd(color ,00000FF00x),8), 0FFx);
  blue=bAnd(bAnd(color ,0000000FFx), 0FFx);

  put color= hex8. alpha= hex2. red= hex2. green= hex2. blue= hex2.;
run;
```

系统输出：color=FF998877 alpha=FF red=99 green=88 blue=77

3.位操作应用——随机数生成

在计算机上生成随机数是一个非常古老的话题，其中用线性反馈移位寄存器（Linear Feedback Shift Register, LFSR）来获得一个给定初始状态的伪随机数序列是个很好的随机数生成机制。一个 n 阶的 LFSR 是由 n 个触发器和若干异或门组成，如图 10-2 所示为一个标准的 16 位斐波那契 LFSR，其中影响下一状态的比特位在 16,14,13,11 处，可表示为 [16,14,13,11,0]（它存在镜像序列为 [16,5,3,2,0]），其反馈多项式为 $x^{16}+x^{14}+x^{13}+x^{11}+x^{0}$。当且当该多项式仅为本原多项式时，寄存器循环可以获得除 0 外的最大长度 2^n-1 个状态，如 $2^{16}-1=65535$。

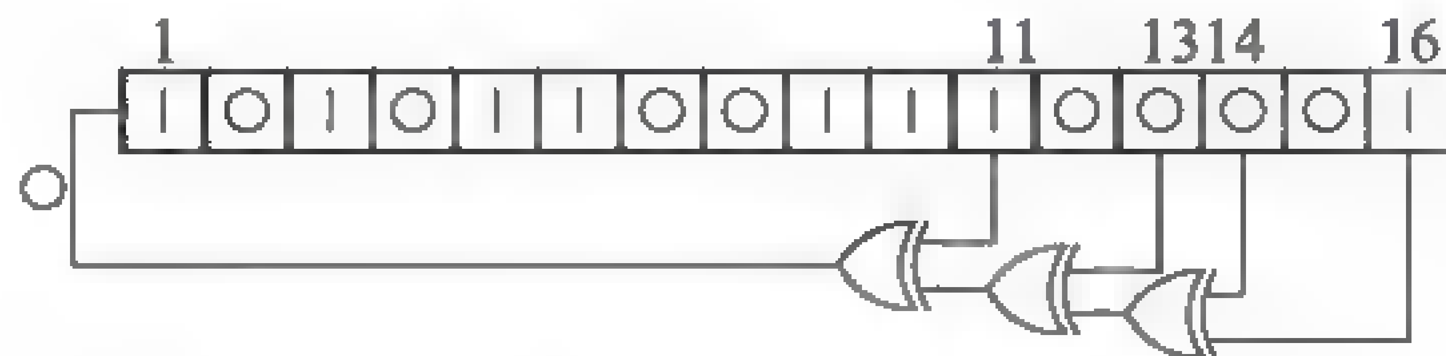


图 10-2　标准的斐波那契 LFSR

寄存器的初始状态被称为种子，每次寄存器生成新的数据都是基于其当前状态经过线性变换所得。本原多项式通过“按位 XOR 运算”这一常见的单比特线性函数和移位操作，就能生成循环周期非常长的随机序列。程序 10-8 为基于此思想实现的随机数生成器。

程序10-8 基于斐波那契LFSR 的随机数生成器

```
proc fcmp outlib=work.prob.random;
  function RandLFSR(lfsr);
    outargs lfsr;

    bit=bAnd(
      bXOR(
        bXOR(
          bXOR(
            bRSHIFT( lfsr, 0),
            bRSHIFT( lfsr, 2)
          ) ,
          bRSHIFT( lfsr, 3)
        ),
        bRSHIFT( lfsr, 5)
      ),
      1);
    lfsr=bXOR(
      bRSHIFT(lfsr,1),
      BLSHIFT(bit, 15));
    return(lfsr);
  endsub;
run;

options cmplib=work.prob;
data _null_;
  x=44257; /*大于 0 的种子数, 即十六进制的 0ACE1x;*/
  do i=1 to 15;
    x=randLFSR(x);
    put x @@;
  end;
run;
```

系统输出前 15 个伪随机数如下:

```
22128 43832 21916 10958 5479 35507 17753 8876 37206 51371 58453 29226 14613
7306 36421
```

4.位操作应用——枚举子集

给定一个集合 S，如果我们要枚举该集合的所有可能的子集。假定集合个数为 N，则 0 到 2^{N-1} 个整数刚好可以映射该集合的所有子集。比如，集合 $\{a, b, c\}$，$N=3$，则 0 到 7（$=2^3-1$）共 8 个整数的二进制值为 000，001，010，011，100，101，110，111。假设 1-3 位分别对应 a、b、c 三个字母，如果特定位为 1 则表示该元素在子集中，共可得 8 个子集：{ }，$\{a\}$，$\{b\}$，$\{a,b\}$，$\{c\}$，$\{a,c\}$，$\{b,c\}$，$\{a,b,c\}$ 这些子集正是该集合所有可能的子集，其代码实现如程序 10-9 所示。

程序10-9 枚举集合的所有可能子集

```
data _null_;
  S= "abc" ;
  length subset $ 4 ; /*length(S)+1*/

  n=length(S);
  do i=0 to bLShift(1,n )-1;
    subset=" ";
    do j=0 to n-1;
      if bAnd( i , bLShift(1,j))^=0 then do;
```

```
            e=substr(S,j+1,1);
            subset=trim(subset) || trim(e) ;
          end;
       end;
       put "subset[ " i "]="  subset;
    end;
run;
```

系统输出：

```
subset[ 0 ]=
subset[ 1 ]=a
subset[ 2 ]=b
subset[ 3 ]=ab
subset[ 4 ]=c
subset[ 5 ]=ac
subset[ 6 ]=bc
subset[ 7 ]=abc
```

枚举子集可用于生成可能的口令组合，在某些网络入侵和软件破解时非常有用。在一般的数据分析中更多的是利用二进制表示的这一特性，表示一些树状结构的数据序列。

第 11 章

扩展 SAS 功能

动态链接库（Dynamic Link Library）是操作系统级别的可重用指令集合。比如，Windows 平台的 *.DLL 和 Unix 平台的 *.so 文件，它使各种计算机语言在编译后能够彼此调用。用户可以将一种计算机编程语言写的程序，编译成动态链接库后可以被另外一种计算机语言加载和调用，从而实现了代码逻辑高级复用。

尽管 SAS 系统已经具有极其强大的算术、逻辑、日期/时间、字符串、文件管理、统计、财务、经济等各方面函数库，但 SAS 依然保留了编程语言的开放性。比如，我们可以在 SAS 语言环境中调用操作系统 DLL 动态库中已经实现的功能，也可以调用用户以其他计算机语言开发的 DLL 函数库，如自定义的数据加密功能，操作系统环境检测、并发文件读写等。SAS 作为一种面向分析的第四代计算机语言，它并不希望把自己设计的过于复杂，但希望保留足够的扩展性，让用户可以在 SAS 语言中调用任何 Windows API 函数以及用户自定义函数库，本章以实例探讨如何在 SAS 中复用已有的 DLL 函数库以及如何自己编写 DLL 函数库来扩展 SAS 功能。

11.1 通过 Module 调用外部 DLL 函数

要在 SAS 语言中调用外部语言编写的 DLL 函数库，可以在 DATA 步中通过调用 Call Module 例程和 ModuleN、ModuleC 函数实现，也可以在 PROC IML 过程步中调用 Call ModuleI 例程和 ModuleIN 和 ModuleIC 函数实现。

由于 SAS 内部使用浮点数和字符型两种基本数据类型，这些函数在调用外部 DLL 函数时需要有一个由 SASCBTBL 文件引用所指向的文本文件，用于描述 SAS 如何变换参数和函数接口调用。它包括参数列表和返回值等信息，称为 SASCBTBL 属性文件。一个 SASCBTBL 属性文件可以包括一个或多个函数的接口描述信息，其格式示例如下：

```
Routine MessageBoxA module=USER32 minarg=4 maxarg=4
  stackpop=called returns=short;
Arg 1 input num  format=pib4. byvalue; * hWnd;
Arg 2 input char format=$cstr200.;     * lpText ;
Arg 3 input char format=$cstr200.;     * lpCaption ;
Arg 4 input num  format=pib4. byvalue; * Style;
```

从上面的内容可以看出，该描述其实就是函数的签名信息加上 SAS 特定的格式化信

息。该文本描述了在 SAS 中调用标准的 Windows API 系统消息框函数 MessageBoxA 所需接口定义，而该 Windows API 函数等价的 C++ 定义如下：

```
int WINAPI MessageBox(
   In opt  HWND     hWnd,
   In opt  LPCTSTR lpText,
   In opt  LPCTSTR lpCaption,
   In      UINT     uType
);
```

随着计算环境进入 64 位时代，微软公司已经在 MSDN 文档上逐渐统一 Win32 函数定义原型。早些年 MessageBoxA 和 MessageBoxW 分别表示 ANSI 版和 UNICODE 版，而如今在 MSDN 文档上已逐渐统一。完整的 Win32 API 系统函数描述参见：https://msdn.microsoft.com/en-us/library/windows/desktop

现在我们将前面的SASCBTBL 属性文件保存到文本文件 C:\temp\msgbox.source 中。这样我们在 SAS 语言中通过特定文件引用就可以像调用 SAS 系统函数一样方便地调用 Win32 API 函数 MessageBoxA 了。程序 11-1 演示了如何直接将数据行中的描述文本保存到外部文件中：

程序11-1　将数据行中的接口定义写出到外部文本文件中

```
data _null_;
  input;

  file "C:\temp\msgbox.source";
  length line $ 255;
  line=trim(_infile_);
  put line;
  datalines4;
Routine MessageBoxA module=USER32 minarg=4 maxarg=4
  stackpop=called returns=short;
Arg 1 input num  format=pib4. byvalue;    * hWnd;
Arg 2 input char format=$cstr200.;        * lpText ;
Arg 3 input char format=$cstr200.;        * lpCaption ;
Arg 4 input num  format=pib4. byvalue;    * Style;
;;;;
run;
```

在 SAS 代码中调用 DLL 函数，首先需要定义名称为 SASCBTBL 的文件引用，它指向上面创建的文本文件；然后用 MODULE 例程或函数间接调用 DLL 中的 WIN32 API 函数。MODULE 例程或函数在调用前会自动解析文件引用 SASCBTBL 所指向的文件内容。程序 11-2 演示了如何在 SAS 代码运行中显示 MESSAGEBOX 消息框。实际上，Windows 平台上的任何图形界面元素在 SAS 中都是可以自由调用的。

程序11-2　在SAS中使用MODULEN 调用Win32 API 函数，接口定义来自sascbtbl

```
filename sascbtbl 'C:\temp\msgbox.source';

data _null_;
  rc = modulen ( 'MessageBoxA', 0 , '唯有变化，才是永恒！', '标题', 4); /*返回 0-7*/
  array msg[8] $ ("内存溢出", "OK", "取消", "终止", "重试", "忽略", " 是", "否 ");
  put "NOTE:  用户点击 [ " msg[rc+1] "]";
run;
```

在 SAS 中执行上面的程序会弹出 Windows 消息窗口，并且日志中会输出你选择的按钮是哪一个（见图 11-1）。你可以更改最后一个参数的值来获得不同的消息框。这种调用机制几乎让 SAS 语言跟 C/C++一样可以调用操作系统平台上的任何功能，如有需要，用户也可在 SAS 代码运行过程中显示任何 Windows 用户操作界面。

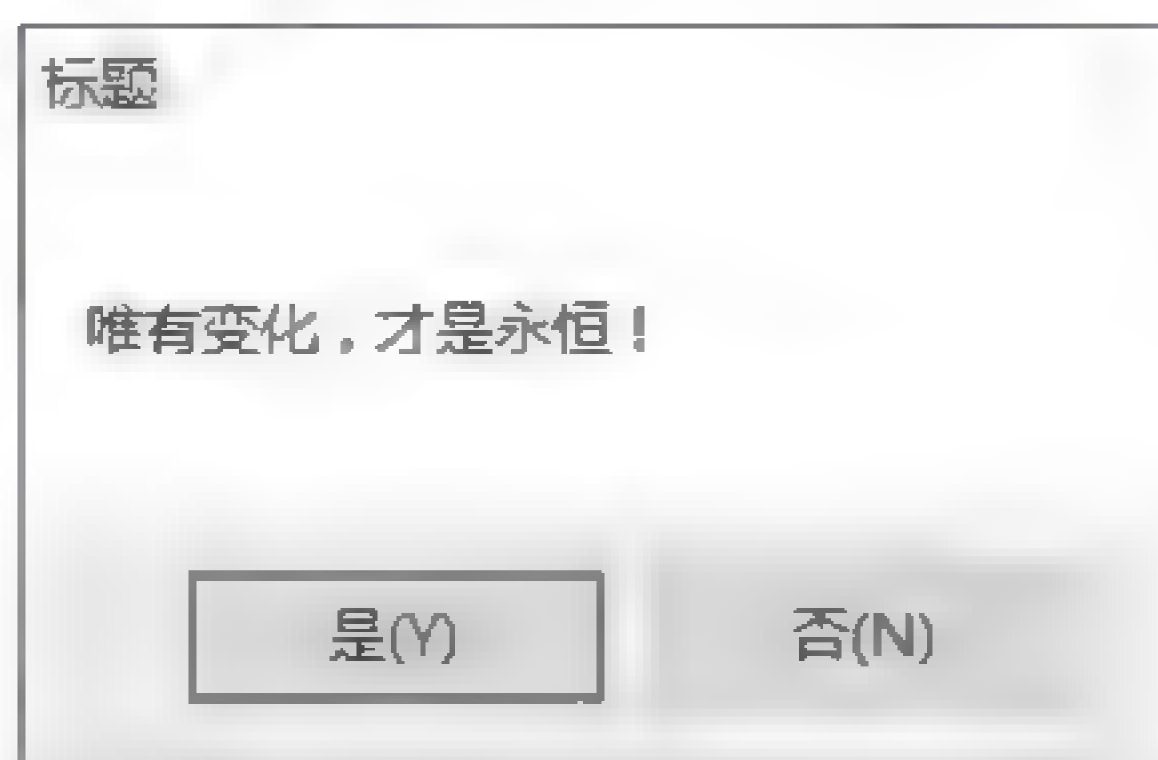


图 11-1　在 SAS 代码中弹出 Windows 消息窗口

与上面的 SAS 代码等价的 C 语言调用如下所示：

```
int msgboxID =MessageBox(NULL,(LPCWSTR)L"唯有变化，才是永恒！"
                         ,(LPCWSTR)L" 标题",4);
```

同理，也可在 SAS 宏代码里实现调用 Windows API 函数，程序 11-3 可获得同样结果。

程序11-3　在 SAS宏中调用Win32 API函数

```
filename sascbtbl 'C:\temp\msgbox.source';
%let rc = %sysfunc (modulen(MessageBoxA, 0, %str(唯有变化，才是永恒！), %str(标题), 4));
```

文件引用 SASCBTBL 除了指向一个外部的 *.SOURCE 文件外，也可以将它指向一个 SAS 内部的 CATALOG，CATALOG 是存储在 SAS 逻辑库内的一种存储格式，用户只需要将代码：

```
filename sascbtbl 'C:\temp\msgbox.source';
```

更改为

```
filename sascbtbl catalog 'work.winapi.msgbox.source';
```

此时引用的文件保存在临时逻辑库 WORK.WINAPI 中，为了让 SAS 代码和调用接口定义存于同一 SAS 代码文件，通常采用如下写法，这样就不必维护额外的外部文件（见程序 11-4）。

程序11-4　将数据行中的接口定义写入SASCBTBL 然后直接调用Win32 API函数

```
filename sascbtbl catalog 'work.winapi.msgbox.source';

/*函数调用接口定义*/
data _null_;
  file sascbtbl;
  input;
  put _infile_;
  datalines4;
```

```
Routine MessageBoxA module=USER32 minarg=4 maxarg=4
stackpop=called returns=short;
Arg 1 input num format=pib4. byvalue; * hWnd;
Arg 2 input char format=$cstr200.;     * lpText;
Arg 3 input char format=$cstr200.;     * lpCaption;
Arg 4 input num format=pib4. byvalue; * Style;
;;;;
run;

/*函数调用*/
data _null_;
  rc = modulen ( 'MessageBoxA', 0 , '唯有变化，才是永恒！' , '标题', 3);
  put rc=;
run;
```

如果用户没有指定 SASCBTBL 文件引用，SAS 的 MODULE 例程或函数不会对参数做任何调整，而是尝试直接加载 DLL 并调用目标函数。然而这种调用具有很高的风险，因为它涉及操作系统层的指针处理和内存访问，就跟在 C C++ 语言中调用操作系统库是一样的道理。此时处理不当可能会引起程序崩溃甚至计算机重启，所以笔者只建议有 Windows 开发经验的高级 SAS 编程人员使用这种扩展技术。

- SASCBTBL语法定义

表 11-1 列出了 SAS 系统提供的 6 个 Module 例程或函数，分别用在 DATA 步和 PROC IML 过程步中调用。

表 11-1　DATA 步与 SAS/IML 可用的模块函数

使用场景	常规参数	包含向量或矩阵参数（PROC IML 专用）
没有返回值时使用	CALL MODULE	CALL MODULEI
返回值为数值类型时使用	MODULEN	MODULEIN
返回值为字符类型时使用	MODULEC	MODULEIC

SASCBTBL 接口描述文件主要包括两部分内容：方法定义和参数列表（见表 11-2）。它是一种抽象的函数签名描述，仅在调用上面的 6 个例程或函数时才用。调用模板文件中也可使用 SAS 行来注释。

表 11-2　SASCBTBL 接口描述文件说明

ROUTINE **MethodName**	**定义函数名称，大小写敏感**
<MINARG=minarg>	允许的最小参数个数
<MAXARG=maxarg>	允许的最大参数个数
<CALLSEQ=BYADDR | BYVALUE>	调用方式：按地址 | 按值传递参数
<STACKORDER=R2L | L2R>	参数压栈顺序：从右到左 | 从左到右
<STACKPOP=CALLER | CALLED>	弹栈方式：由调用者 | 被调用者弹栈
<TRANSPOSE=YES | NO>	是否转置矩阵
<MODULE=name>	**源 DLL 动态库名称**，系统路径中自动搜索
<ARCH=16/32>	体系结构：16 位 | 32 位体系结构
<DLLTYPE=32 | 64 | X32 | X64>	动态库类型：32|64，仅限于 Windows
<RETURNS=SHORT | USHORT | LONG | ULONG | INT64 | DOUBLE | DBLPTR | CHAR<n>>	返回值类型，共 8 种
<RETURNREGS=DXAX>;	返回值寄存器

（续表）

ARG **ArgumentNumber**	**参数的编号**，从 1 开始，不需要名称
<INPUT | OUTPUT | UPDATE>	访问类型：输入参数 | 输出参数 | 可写参数
<NUM | CHAR>	数据类型：数值 | 字符类型
<BYADDR | BYVALUE>	参数传递方式：按地址 / 按值传递 指针变量应使用按地址传递
<FDSTART>	指示本参数开始一个值块，后续参数为该结构体的一部分
<REQUIRED | NOTREQD>	指示参数是必选还是可选
<FORMAT=format>;	处理时的输入输出格式
* Comments in SASCBTBL;	可以包括以 * 开头，分号结束的行注释

参数中如果出现缺失值（不管是任何形式的缺失值 . 还是 ._ 或 .A），它们在调用前都会被自动转换成 0 再进行调用。

由于 SAS DATA 步中只有数值和字符两种数据类型，因此我们需要使用各种 SAS 格式化功能来对数据作变换，从而符合函数调用的规则。表 11-3 列出了 C 语言数据类型和对应 SAS 输入输出格式之间的映射关系。

表 11-3　C 语言数据类型和对应 SAS 输入输出格式之间的映射关系

C 语言数据类型	SAS 输入输出格式
signed int signed long	IB4.
signed short	IB2.
float	RB4. FLOAT4.
double	RB8.
unsigned int unsigned long	PIB4.
unsigned short	PIB2.
char *	IB4.（32bit SAS） IB8.（64bit SAS）
char[w] 长度为 w 的字符串	$CHARw.
\0 结尾的字符串，最大 w 个字符	$CSTRw.
32- 位或 64- 位的指针，取决于 OS	$PTR.
char 当作 short 按值传递	$BYVAL2.
char 当作 long 按值传递	$BYVAL4.
char 当作 double 按值传递	$BYVAL8.

其中任何不以 $ 开头的格式都是数值变量或表达式，但如果数值变量遇到 $ 开头的格式，如 $CHAR，则数值将会被用 BEST. 格式自动转化为字符串（长度默认由 BEST. 格式决定）进行调用，调用后改变的字符串会再按照标准的 INFORMAT 转换回数值。

如果传递给 MODULE 例程或函数的是字符类型，而参数定义中指定了某个数值格式，则字符串使用标准输入格式转换为一个数值；调用返回时数值再被按照长度为字符参数长度的 BEST. 格式转化回字符串。

另外，在调用 Win32 API 函数时，会用到微软 MSDN 文档中定义的一系列 Microsoft 特定数据类型，如 HKEY、LPCSTR、DWORD、REGSAM,、PHKEY 等，可以查看 MSDN 文档映射到标准 C 语言数据类型，结构体或指针类型上，再参照表 11-3 来指定恰当的转化格式。

11.2 用 C 语言开发用户函数库

如果我们调用的函数不是 Win32 API，而是自己用 C/C++ 语言编写的代码，则 C/C++ 代码必须编译为特定格式的动态库（DLL）。鉴于越来越多的硬件机器是 64 位的环境且 SAS 安装在 64 位 Windows 环境上，下面以 64 位的运行环境为例展示如何用 C 语言开发自定义 DLL 函数库并在 64 位的 SAS 环境中调用它，这样就能根据分析业务需要，使用 C/C++ 扩展计算逻辑和算法，然后在 SAS 代码中调用它。

11.2.1 准备64位 C 编译环境

首先我们需要有能够编译 C 语言为 64 位 DLL 动态库的开发环境，其准备工作如下。

（1）安装 64 位的 GCC 编译器：由于默认 minGW 只提供 32 位版本供安装和使用，可以从 http://tdm-gcc.tdragon.net/download 下载 tdm64-gcc-5.1.0-2.exe 来建立 64 位 Windows 环境的 GCC 编译环境。使用默认选项运行 tdm64-gcc-5.1.0-2.exe 将安装环境到 C:\TDM-GCC-64 路径下，并创建开始菜单快捷方式：

```
C:\ProgramData\Microsoft\Windows\StartMenu\Programs\TDM-GCC-64
  \MinGW Command Prompt
```

（2）点击上面的快捷方式，或运行如下 DOS 命令即可进入 tdm64-gcc-5.1.0-2 编译环境。

```
C:\>%comspec% /k ""C:\TDM-GCC-64\mingwvars.bat""
```

（3）为了测试 GCC 编译环境，可以创建如下 HelloWorld.c 文件并保存到 C:\temp 目录（见程序 11-5）。

程序11-5 HelloWorld.c

```
#include "stdio.h"

int main(){
  printf("Hello,world!\n");
}
```

（4）在 GCC 编译命令行窗口中切换当前路径到 C:\temp，并执行如下编译命令。

```
C:\>cd C:\Temp
C:\Temp>gcc helloworld.c -o helloworld.exe
```

如果能成功编译出可执行文件 helloworld.exe 且执行它能输出"Hello World!"，则说明 GCC 编译环境安装成功，后面可用该编译器来生成 64 位的 DLL 动态库。

11.2.2　开发用户自定义动态库

（1）创建用户自定义 C 代码：为演示目的我们只编写 CubeSum 函数用来计算两个双精度浮点数的立方和 x^3+y^3（见程序 11-6），实际项目中会利用 C 语言开发一些较为复杂耗时的计算和分析逻辑。

程序11-6　CubeSum.C

```
#include <windows.h>
#include <math.h>

double __stdcall cubesum(double a, double b) {
  return pow(a, 3) + pow(b, 3) ;
}
```

有两点需要特别注意：①如果引用了其他 C 语言的头文件，需要将它一并引用，如 pow() 函数包含在 math.h 头文件中；②在需要导出的函数前面添加方法修饰 __stdcall 。

（2）执行如下编译命令，将源文件 CubeSum.c 编译成标准可调用的动态库 CubeSum.dll

```
gcc cubesum.c -o cubesum.dll -m64 -shared
```

编译成功后，可以使用 C:\Windows\System32\Rundll32.exe 做简单的测试，检查我们的 DLL 是否编译完整。如果命令执行报错说明动态库的依赖没找到或者存在其他问题。检测命令为

```
C:\Windows\System32\rundll32.exe cubesum.dll,cubesum  1 2
```

（3）为了让编译的动态库能够被操作系统上的其他程序调用，通常需要把 DLL 放到 Windows 系统环境变量 PATH 所指定的路径中，最典型的做法是把 DLL 文件放到系统目录 C:\Windows\System32 中，这样 DLL 就可以被该计算机上任何目录中的应用或 SAS 程序调用。

注意：在某些 Windows 10 系统上用 DOS 命令行复制它到 C:\Windows\System32 会失败，因为该环境上操作系统会显式要求你具有管理员权限才可更改 C:\Windows\System32 目录中的内容。

（4）为了让 C 语言编译的动态库能够被 SAS 程序调用，同样需要生成 SASCBTBL 属性文件。该文件描述了每一个需要被 SAS 调用的函数接口定义，源 DLL 和参数序列描述，其本质是调用接口函数签名。CubeSum 函数的调用接口定义如程序 11-7 所示。

程序11-7 C函数CubeSum 的接口定义

```
data  null ;
  file sascbtbl;
  input;
  put  infile ;
  datalines4;
Routine cubesum minarg=2 maxarg=2 stackpop=called
  module= cubesum returns = double;
Arg 1 num byvalue format=rb8.; arg 2 num byvalue format=rb8.;
;;;;
run;
```

（5）如果 SASCBTBL 属性文件生成成功，则可以在 SAS 的数据步或者 PROC IML 里面调用该 CUBESUM.DLL 动态库中的函数（见程序 11-8）。调用的时候需要指定同样的 SASCBTBL 属性文件位置，如果调用模板定义在 SAS CATALOG 中，调用方也要使用与之匹配的 CATALOG 路径。

程序11-8 调用 CubeSum函数的 SAS 程序示例

```
filename sascbtbl catalog "work.interop.cubesum.source";
data _null_; x = 3;
  y = 1;
  z = modulen( 'cubesum', x,y);
  put "NOTE: CubSum( " x "," y " )=" z;
run; quit;
```

如果成功调用，在 SAS 日志中应该会输出：

```
NOTE: CubSum(3,1)=28
```

11.3 PROTO 编写 C 代码或注册外部 DLL

自从 SAS 9.2 引入 PROC FCMP 过程步后，用户可以用 SAS 语言来编写用户自定义函数或例程，编写的函数/例程可以广泛用于 SAS DATA 步，这是 SAS 语言的里程碑式的进步。然而 SAS 走得更远，它提供 PROC PROTO 过程步允许用户在 SAS 语言中直接用 C 语言和数据类型编写的 C 函数，这些 C 函数一旦注册后就可直接在 SAS 的 PROC FCMP 和 PROC COMPILE 过程步的 SAS 函数或例程中调用。

用户在 PROC PROTO 里直接编写 C 语言代码，运行 SAS 时那些源代码会自动编译并注册到 SAS 运行环境中，此时那些代码也可在 FCMP 函数或例程中随意调用。尽管 SAS DATA 步不能直接调用 PROC PROTO 中用 C 语言编写的那些函数，但通过 PROC FCMP 函数的包装桥接，SAS 数据步内也可以非常方便地像调用 SAS 系统函数一样调用 PROC PROTO 中那些 C 语言编写的函数。

PROC PROTO 也支持从外部动态库中加载机器指令代码，这是 SAS 程序里调用外部动态库的另外一种方法。假定 DLL 文件已被复制到系统路径 C:\Windows\System32 中，下面的代码（见程序 11-9）将加载我们自己编写的 DLL 文件到当前 SAS 运行环境中。

程序11-9　使用PROC PROTO 链接到外部动态库

```
proc proto package work.proto.cubesum stdcall;
  link "C:\Windows\System32\cubesum.dll";
  double cubesum(double, double);
run;
```

在前面生成 DLL 函数库时我们指定了 stdcall 调用规范，而 stdcall 仅在 PC 平台上支持，更一般地如果编译一个 C 函数并且希望能够从 DLL 导出给别人使用，需要在函数声明时指定它。可在开发 C 语言函数 CUBESUM 时用 __declspec（dllexport）去声明函数，比如：

__declspec（dllexport） double cubesum（double, double）;

实际上，对于一些简单的分析项目和逻辑，完全不必要用 DLL 库进行包装，而是在 SAS 程序里直接用 C 语言编写分析逻辑即可。比如，前面的 CubeSum 函数，可直接在 PROC PROTO 里用 C 语言直接编写源代码（见程序 11-10）。

程序11-10　PROC PROTO 直接用C 语言实现计算函数（斜体部分为C 语言代码）

```
proc proto package=work.proto.cubesum;
  double cubesum(double, double);
  externc cubesum;
    double cubesum(double x, double y)
    {
      return pow(x,3)+ pow(y,3);
    }
  externcend;
run;
```

为了能在 SAS 数据步里调用自定义的 CUBESUM 函数，需使用 PROC FCMP 来包装桥接 PROTO 包——用 INLIB 选项来指定被引用的包，并定义一个 CUBESUM_FCMP 函数（见程序 11-11）。

程序11-11　FCMP 函数封装PROTO 中链接的外部函数，以供DATA 步使用

```
proc fcmp inlib=work.proto outlib=work.funcs.cubesum;
  function cubesum_fcmp(x, y);
    rc=cubesum(x, y); return(rc);
  endsub;
run;
quit;
```

运行上面代码后，cubesum_fcmp 函数就已经可用了，此时我们就可在 DATA 步里随便调用该 PROTO 包中的 C 函数了（见程序 11-12）。

程序11-12　调用FCMP->PROTO->DLL 函数示例

```
options cmplib=(work.proto work.funcs);
data _null_;
  x=300; y=100;
  z=cubesum_fcmp(x,y); put z=;
run;
```

系统输出结果：z=28000000

PROC PROTO 创建的 C 函数经过 PROC FCMP 桥接后，C 函数可在 DATA 步中进行调用。这种扩展功能将在后面章节中探讨如何使用指针构建复杂数据结构时会用到。

小知识：SAS 是世界上少数拥有 C 编译器核心技术的软件公司，20 世纪 80 年代 SAS 公司为 IBM 主机系统重写 SAS 系统时决定自己开发 C 编译器；1985 年 11 月 SAS 公司发布了第一个 C 编译器 2.10C 版本，并在 1986 年收购了 PC 机 C 编译器公司 Lattice。SAS 的编译器支持 IBM 大型机和 IBM 个人电脑交叉编译，它让 C 语言编写的代码能够运行在 VM/CMS、MVS 和 MVS/XA 上；SAS 的 MVS 编译器技术甚至比 IBM 的 WhiteSmith C 编译器还好。SAS 公司私有的 SAS/C 编译器为 SAS 系统强大的技术实力提供了保障。所有的 SAS 软件核心技术代码和过程步都是基于 SAS 自己的 C 编译器编译，它保证了 SAS 软件异乎寻常的跨平台迁移性和卓越的系统性能。SAS/C 编写的代码能够运行在多达十余种主流的 Unix/Linux 系统和 Windows/MVS 操作系统之上，主要原因之一就是 SAS 拥有自己的核心编译技术。

第 12 章

数据结构——数组

在计算机程序员眼中，数据结构和算法是构成编程世界的基础。除了常量和变量之外，我们无法得知全能的上帝是否也曾用过数组、链表、队列、二叉树、矩阵、有向图、无向图之类的数据结构对这个世界进行编程。实际上，数据结构只是一种信息的组织和表达，是为了呈现程序世界中变量或对象之间的逻辑关系而构建的信息组织单元。不可否认的一点是，计算机科学中的数据结构和算法几乎可以完美表达现实世界中的任何现象或事物的关系。任何复杂的问题都可通过恰当的数学抽象，然后通过数据结构和算法在计算机上得以重构和求解。

本章将站在程序员的角度思考如何在 SAS 中实现传统计算机科学中的基础数据结构，首先回顾一下由相同类型变量组成的变量集合，也是最早出现和最重要的复合数据结构“数组”。

12.1 数组

基本数据类型之外，最常用的数据结构为数组，它就是通过数组名和索引（或者称为下标）对一系列的变量进行引用的数据结构，如 A[1]、B[1,2]。与 C C++ 不同，SAS 数组的第一个元素通常由下标为 1 开始索引，但用户也可以改变初始下标使它从 0 开始。根据维度数量可以将数组分成一维数组和多维数组，实践中用得最多的是二维数组。在高等数学计算和数值分析中，二维数组通常和数学上的矩阵概念相关。掌握数组以及它和 SAS 数据集之间的转换是在 SAS 中进行数值计算和分析的基础。

从前面的学习中可以看到，SAS 语言提供了传统 DATA 步、SAS 宏、FCMP 以及 DS2 等基础子系统。抽象的数组在不同的子系统中具有不同实现和语法，但它们都是简单的线性数据结构。本章主要关注数组这种数据结构与 SAS 数据集之间的转换，以及数组在各子系统中应用对比；最后举例说明数组的高级应用。

12.1.1 DATA步数组

DATA 步数组包括变量数组和临时数组，它们都使用 ARRAY 语句创建，但临时数组需要用 TEMPORARY 修饰且不需要变量映射，默认不输出到外部数据集。程序 12-1 展示了如何从 DATA 步数据行获取数据构建一维数组并计算每行数据的和，结果如图 12-1 所示。

程序12-1 从数据行读取数据到变量数组

```
data d;
  input  c1-c3;       /*读入 datalines 中的内嵌数据*/
  array c[3];         /*定义 SAS 数组，默认映射到变量c1-c3*/
  total=sum(of c:);
  put c[*]= total=;   /* 一维数组 c 就对应每一行记录*/
  datalines;
1 2 3
4 5 6
;

proc print data=d;
  title 'var array';
run;
```

Obs	c1	c2	c3	total
1	1	2	3	6
2	4	5	6	15

图 12-1 DATA 变量数组

临时数组没有变量映射，默认不会被输出到外部数据集，使用方法与变量数组基本相同。但它与变量数组有细微的不同：临时数组不能使用 put a[*]=; 进行输出。下面的例子展示了从数据行构建二维临时数组的简洁用法（见程序 12-2）。

程序12-2 从数据行读取数据到临时数组

```
data _null_;
  array a [2,3] _temporary_; /*临时数组没有变量映射*/
  do i = 1 to dim(a);
    do j=1 to dim(a,2);
      input x @@;
      a[i,j] = x;
      put a[i,j]= @@; /*打印到日志*/
    end;
    put;
  end;
datalines;
1 2 3
4 5 6
run;
```

系统输出：

```
a[1,1]=1 a[1,2]=2 a[1,3]=3
a[2,1]=4 a[2,2]=5 a[2,3]=6
```

除了从数据行获得数据，是否有办法直接从外部数据集读取数据到 DATA 步中的二维数组呢？答案是肯定的。下面的例子先创建一个演示数据集，然后在另一 DATA 步中基于它构建数组用于分析计算（见程序 12-3）。

程序12-3 将数据行中的数据读到二维数组供计算分析之用

```
data d1;
  input c1-c3 @@;
  datalines;
```

```
1 2 3
4 5 6
;
data _null_ ;
  array a[3] c1-c3;/*映射*/
  array b[2,3] _temporary_ ;
  do i=1 to last;
    set d1 point=i nobs=last;
    do j=1 to dim(b,2); /*一维数组赋给二维数组*/
      b[i,j]=a[j];
    end;
  end;

  do i = 1 to dim(b);/*打印二维数组，结果同前*/
    put b[i,1]= b[i,2]= b[i,3]=;
  end;

  stop;
run;
```

12.1.2　FCMP 数组

运行在 PROC FCMP 子系统中，该子系统提供 *READ_ARRAY* 和 *WRITE_ARRAY* 系统函数实现外部 SAS 数据集和 FCMP 数组之间的自由转换，方便我们编写各种统计处理算法。程序 12-4 展示了如何在 FCMP 中利用二维数组对数据集所有数据遍历求和。

```
程序12-4  读入 SAS数据集到FCMP 二维数组
data d;
  input c1 c2 c3;
  datalines;
1 2 3
4 5 6
;
run;

proc fcmp;
  array a[2,3] / nosymbols;
  rc=read_array('d',a); /*指定源数据集 d 和目标二维数组 a*/
  /* 可定制读入顺序和个数，如 rc=read_array('d', a, 'c1', 'c3');*/

  sum=0;
  do i=1 to dim(a);
    do j=1 to dim(a,2);
      sum=sum+a[i,j];
    end;
  end;
  put sum=;
run;
quit;
```

运行代码，系统输出 sum=21

FCMP 数组也可方便地用 WRITE_ARRAY 函数导出 SAS 数据集（见程序 12-5），供其他过程步或分析代码进一步使用，结果如图 12-2 所示。

程序12-5 将FCMP 二维数组写入SAS 数据集

```
proc fcmp;
  array x[2,3] (1 2 3 4 5 6);
  rc=write_array('work.fcmp_array', x);
  /*也可指定列名*/
  /*rc=write_array('work.fcmp_array', x, 'c1', 'c2', 'c3');*/
run;
proc print data=work.fcmp_array;
run;
```

Obs	x1	x2	x3
1	1	2	3
2	4	5	6

图 12-2 导出 FCMP 数组到数据集

更多的时候，会利用 FCMP 函数编写可重用的计算逻辑，比如根据数学定义实现平均绝对偏差函数 AVEDEV。统计学上，偏差表示每个数值与平均值之间的差值，而平均偏差表示每个偏差绝对值的平均值，AVEDEV 函数可用来衡量数据离散程度（见程序 12-6）。

程序12-6 利用数组作为FCMP函数参数进行计算

```
data d;
  input x @@;
  datalines;
1 2 3 4 5 6
;
proc fcmp outlib=work.funcs.math;
  function avedev(d[*]);/*定义输入数组参数，多维数组也相同*/
    length=dim(d);
    sum=0;
    do i=1 to length;
      sum += d[i];
    end;
    mean=sum/length;/*计算平均值*/

    sumdev=0;
    do i=1 to length;
      sumdev += abs( d[i]-mean );
    end;
    avedev=sumdev/length;/*计算偏差绝对值总和与平均值*/
    return(avedev);
  endsub;

  /* 演示程序 */
  array a[6] / nosymbols;
  rc=read_array('d',a);
  avedev = avedev (a);
  put avedev=;
run;
quit;
```

系统输出：avedev=1.5

更多时候是在 DATA 步内调用自定义函数，此时函数中输出的结果会被输出到 SAS 日志窗口，而不是输出窗口（见程序 12-7）。

程序12-7　在DATA 步内构造数组传递给FCMP函数进行处理

```
options cmplib=work.funcs;
data _null_ ;
  array a[6] _temporary_ ;
  do i=1 to last;
    set d point=i nobs=last;
    a[i]=x ;
  end;
  avedev = avedev (a);
  put avedev=;
  stop;
run;
```

在计算过程中，有时需要动态创建临时数组用于存放计算的中间结果，然而在编写代码时数组长度并不确定，因此需要借助 CALL DYNAMIC_ARRAY 例程来扩展数组长度，实现复杂的统计学计算过程。比如，下面的例子实现了 WACKY 平均标准偏差的计算逻辑，其中临时数组 t[ ] 存放了上一个观测的值用于后续计算（见程序 12-8）。

程序12-8　在FCMP 函数中使用动态数组

```
proc fcmp outlib=work.funcs.math;
  function avedev_wacky(d[*]);
    length=dim(d);
    array t[1] /nosymbols; /*定义临时数组 t 然后扩展长度为 length*/
    call dynamic_array(t, length);


    sum=0;
    do i=1 to length;
      sum += d[i];
      if i>1 then t[i]=d[i-1];/*给临时数组赋值*/
      else t[i]=0;
    end;
    mean=sum/length;

    sumdev=0;
    do i=1 to length;
      sumdev += abs( (d[i])-t[i] /2-mean );
    end;
    avedev=sumdev/length;
    return(avedev);
  endsub;
run;
quit;
/*调用程序示例*/
options cmplib=work.funcs;
data d;
  array a[6];
  do i=1 to 6;
    input x @@;
    a[i]=x;
    keep x;
    output;
  end;
  avedev_wacky = avedev_wacky (a);
  put avedev_wacky=;
```

```
  datalines;
1 2 3 4 5 6
;
```

系统输出：avedev wacky=1.25

12.1.3 DS2 数组

DS2 是传统 DATA 步的强化与扩展，它不但有更强的数据类型支持和作用域概念，而且它支持 2 种不同类型的数组：变量数组与标准数组。其中变量数组与 DATA 步的数组相同，程序 12-9 展示了 DS2 变量数组的使用，其输出数据集 MYDATA 中的最后一列包含每行数据的总和（结果如图 12-3 所示）。

程序12-9 DS2 中的变量数组，默认会输出到数据集中

```
data d;
  input c1 c2 c3;
  datalines;
1 2 3
4 5 6
;

proc ds2;
  data mydata(overwrite=yes);
    vararray double c[3]; /*使用 vararray 映射 c1,c2,c3*/
    dcl double sum;
    method run();
      set d;
      sum=sum(c[1], c[2], c[3]); /*一维数组 va 映射数据集每行*/
    end;
  enddata;
run;
quit;
proc print data=mydata;run;
```

Obs	c1	c2	c3	sum
1	1	2	3	6
2	4	5	6	15

图 12-3 DS2 变量数组生成数据集

DS2 标准数组与 C/C++ 的数组概念相同，程序 12-10 综合展示了如何利用 DS2，计算每列数据之和并将它们输出到数据集 MYDATA 的最后一行中（结果如图 12-4 所示）。

程序12-10 DS2 中的标准数组，默认不会输出到数据集

```
proc ds2;
  data mydata(overwrite=yes);
    dcl double s[3];
    method run();
      set d;
      s[1]+c1; s[2]+c2; s[3]+c3; /*每列求和*/
```

```
      output;
    end;

    method term();
      c1=s[1]; c2=s[2]; c3=s[3]; /*输出和值*/
      output;
    end;
  enddata;
run;
quit;
proc print data=mydata;
  title 'std array';
run;
```

Obs	c1	c2	c3
1	1	2	3
2	4	5	6
3	5	7	9

图 12-4　DS2 标准数组生成数据集

需要特别注意一点，变量数组会自动映射 PDV 变量并输出，而标准数组则不会。程序 12-9 输出结果中最后一列是每行数据的和，而程序 12-10 输出结果的最后一行是输入数据集每列的和。

12.1.4　SAS宏数组

在 SAS 宏系统中没有所谓数组的概念，但这不妨碍利用宏自身的特性构建宏数组。阿基米德说过“给我一个支点，我可以撬起整个地球”。这里要说的是，给我们一个变量，我们可以撬起整个数据结构！下面的SAS宏代码构建出一个拥有10个元素的“宏数组”，并对其赋值和打印（见程序 12-11）。

程序12-11　利用多重解析在SAS宏中模拟一维数组

```
%macro MVAR_ARRAY;
  %do i=1 %to 5;
    %local a&i;
    %let a&i=&i ;
    %put a[&i]=&&a&i;/*双重转义*/
  %end;
%mend;
%MVAR_ARRAY;
```

系统输出：

```
a[1]=1
a[2]=2
a[3]=3
a[4]=4
a[5]=5
```

是否能实现 SAS 宏版的“二维数组”？答案也是肯定的。这种方法可用于通过 SAS 宏模拟数组运算或利用 SAS 宏在 DATA 步和 PROC 步之间交换数据的场景（见程序 12-12）。

程序12-12　利用多重解析在SAS宏中模拟二维数组

```
%macro MVAR_2DARRAY;
  %do i=1 %to 2;
    %do j=1 %to 2;
      %local a&i._&j;
      %let a&i._&j= %eval( (&i-1)*3 + &j );
      %put a[&i,&j]=&&a&i._&j; /*双重转义*/
    %end;
  %end;
%mend;
%MVAR_2DARRAY;
```

系统输出如下：

```
a[1,1]=1
a[1,2]=2
a[2,1]=4
a[2,2]=5
```

12.2　数组应用：高精度数值计算

数组作为一种元素数据类型相同的简单线性结构，它的用途非常广泛。其中，一个特殊用途是利用整个数组的所有元素来表示一个所谓的“大数”。大部分编程语言的数值数据类型，由于采用固定字节长度的存储结构，它在计算过程中可能因数据溢出有效的表达范围导致精度损失，为了保证计算数据精度，可基于数组这一线性结构来构建高精度数值运算框架。

在前面的章节中我们探讨了生成黄金分割数列问题，但我们最多只能够生成 78 位精确表示黄金分割数列，从第 79 位开始实际上已经丢失了部分精度。为了能够生成任意长度的黄金分割数列表示，需要借助数组这一线性结构，将多个数组元素组合表示一个表达范围更加宽广的“大数”。笔者编写的 SAS 程序（见程序 12-3）可生成 2048 位的无精度损失的黄金分割数列，读者可修改参数生成任意位数的精确黄金分割数列。

程序12-13　突破 8字节存储限制的黄金数列生成器，单个数组元素表示5 位数字

```
proc fcmp outlib=work.funcs.math;
  function add(a[*], b[*], c[*] );
    outargs c;
    carry=0;
    do i=1 to dim(c);
      c[i]=a[i]+ b[i]+ carry;
      if c[i]< 1E5 then carry=0;
      else do;
        c[i]=c[i]-1E5;
        carry=1;
      end;
    end;
```

```
    return(carry);
  endsub;
run;

options cmplib=work.funcs;
data Fbnc;
  /*用长度为 256 的数组表示 256*5 位十进制长度的大数 */
  array y1[256]  temporary ;
  array y2[256]  temporary ;
  array y [256]  temporary ;

  do i=1 to dim(y);/*初始化*/
    y1[i]=0;y2[i]=0;y[i]=0;
  end;

  do n=1 to 2048;/*计算 2048 位 FBNC*/
    if n=1 or n=2 then y[1]=1;
    else rc=add(y1,y2,y );

    do i=1 to dim(y);/*保留上两位数*/
      y2[i]=y1[i]; y1[i]=y[i];
    end;

    /*合并结果*/
    length s $ 32767;
    s=" ";
    do i=dim(y) to 1 by -1;
      if y[i]^=0 then do;
        vstart=i;
        leave;
      end;
    end;
    do i=vstart to 1 by -1 ;
      s=trim(left(s)) || trim(left(put(y[i],z5.)));
    end;

    do i=1 to length(s);
      if substr(trim(left(s)),i,1)^= "0" then do;
        s=substr(trim(left(s)),i);
        leave;
      end;
    end;
    keep s;
    output;/*输出计算结果到数据集*/
  end;
run;
proc print data=fbnc;run;
```

运行程序生成的第 2048 个无精度损失的黄金分割数列为如下 428 位的超大整数：

454153044374378942504557144629068920270090826129364442895118239027897145 2509283435684349718034771730433207742075010299663962500640783801879736380774181591579496806948995766259226048959686056348436218766394283482473000979306575217575924408151880646518264800221975557589955655164820646173515138267042115173436029259905997102292769397103720814141099147144935820441851539180551702416940356101455471043375366140283389830736802626841 01

第 13 章

数据结构——队列与堆栈

列表（List）是数组之外最简单的复合数据结构，常用的列表类型包括队列和堆栈。其中队列可分为单向队列、双向队列和循环队列等。本章主要探讨在 SAS 中如何实现可重用的基础数据结构，为编写高级数据分析程序打好基础。

13.1 队列

队列（Queue）在抽象概念上是一种先进先出（First-In-Fist-Out）的线性表（见图 13-1），具体可以通过数组或者链表来实现。队列从前端进行读取，在后端进行写入或添加。我们也可以在初始化队列时设置队列长度，用来控制队列的容量。队列的常用操作包括初始化、入列、出列、遍历、长度查询、是否为空和清空等操作。



图 13-1 队列示意图

在 SAS 中实现队列有多种方法，分别适用于不同场景。笔者在此给出两个 DATA 步中可重用的通用队列实现方式。

13.1.1 函数版实现与示例

基于 FCMP 函数和 SAS 数据集实现的队列（见程序 13-1）。其核心思想是利用 SAS 数据集作为数据持久化的载体，然后利用 FCMP 函数读写数据集来实现队列基本操作。这种实现由于涉及数据集的读写，也就是说，它需要进行磁盘文件读写操作，因此它只适用于性能要求不那么高的场景。

程序13-1 FCMP版的队列实现，存储使用持久化数据集

```
proc fcmp outlib=work.funcs.queue;
  /* 队列 - 初始化：参数为队列名字和容量 */
  function QueueDefine(name $, maxsize);
    array y[1] / nosymbols;
    if exist(name) then do;
      array x[1] / nosymbols;
      rc=read_array(name,x);
      length=dim(x);
```

```
    x[1]=maxsize;
    rc=write_array(name, x);
  end;
  else do;
    if maxsize>0 then y[1]=maxsize;
    else call missing(y[1]);
    rc=write_array(name, y);
  end;
  return (rc);
endsub;

/* 队列 - 入列：将 value 入列 */
function QueueEnqueue(name $, value);
  array y[1] / nosymbols;

  if exist(name) then do;
    array x[1] / nosymbols;
    rc=read_array(name,x);
    length=dim(x);
    maxsize=x[1];

    if maxsize =. | (length-1)< maxsize then do;
      call dynamic_array(y, length+1);
      do i=1 to length;
        y[i]=x[i];
      end;
      y[length+1]=value;
      rc=write_array(name, y);
    end;
    else do;
      put "ERROR:  Queue is full";
      rc=0;
    end;
  end;
  else do;
    call missing(y[1]);
    call dynamic_array(y, 2);
    y[2]=value;
    rc=write_array(name, y);
  end;
  return (rc);
endsub;

/* 队列 - 出列*/
function QueueDequeue(name $, value);
  outargs value;
    array y[1] / nosymbols;

  if exist(name) then do;
    array x[1] / nosymbols;
    rc=read_array(name,x);

    length=dim(x);
    if length > 1 then do;
      call dynamic_array(y, length-1);
      y[1]=x[1]; value=x[2];
      do i=2 to length-1;
        y[i]=x[i+1];
      end;
    end;
    else do;
```

```
      y[1]=x[1];
      put "WARNING:  Queue" name "is EMPTY!";
    end;
    rc=write_array(name, y);
  end;
  else do;
    rc=0;
  end;
  return (rc);
endsub;

/* 队列 - 检测长度 */
function QueueLength(name $);
  array y[1] / nosymbols;

  if exist(name) then do;
    array x[1] / nosymbols;
    rc=read_array(name,x);
    length=dim(x);
    rc=length-1;
  end;
  else do;
    rc=0;
  end;
  return (rc);
endsub;

/* 队列 - 是否为空 */
function QueueIsEmpty(name $);
  rc=0;
  if QueueLength(name)=0 then rc=1;
  return (rc);
endsub;

/* 队列 - 清空 */
function QueueDelete(name $);
  array y[1] / nosymbols;

  if exist(name) then do;
    array x[1] / nosymbols;
    rc=read_array(name,x);
    y[1]=x[1];
    rc=write_array(name, y);
  end;
  else do;
    rc=0;
  end;
  return (rc);
endsub;

/* 队列 - 遍历  */
function QueueDump(name $);
  if exist(name) then do;
    array x[1] / nosymbols;
    rc=read_array(name,x);
    length=dim(x);
    if length > 1 then do;
      do i=1 to length-1;
        value=x[i+1];
        put "NOTE:  " name "[" i "] " value ;
      end;
```

```
        end;
        else do;
        put "WARNING:  Queue" name "is EMPTY!";
      end;
      return(1);
    end;
    return(0);
  endsub;
run;
quit;
```

程序 13-2 演示了如何在 DATA 步中使用该 FCMP 版的队列，这样我们就能在 DATA 步中使用这种数据结构实现所需算法。

程序13-2　FCMP 版队列的应用示例

```
options cmplib=(work.funcs);
data _null_;
  rc=QueueDefine('q', 3);

  do i=1 to 3;
    myvar=i*10;
    rc=QueueEnqueue('q', myvar);
    put "NOTE:  入列 " myvar;
  end;

  q_len=QueueLength('q');
  do i=1 to q_len;
    rc=QueueDequeue('q', myvar);
    if rc=0 then put "NOTE:  出列 " myvar ;
  end;

  rc=QueueDelete('q');
run;
```

系统输出如下：

```
NOTE:  入列 10
NOTE:  入列 20
NOTE:  入列 30
NOTE:  出列 10
NOTE:  出列 20
NOTE:  出列 30
```

13.1.2　宏版实现与示例

基于 SAS 宏函数和 Hash 对象实现的队列（见程序 13-3）。前面的 FCMP 版的队列使用了 SAS 数据集进行持久化，如果我们要追求运行效率并且希望所有队列操作都是在内存中进行，则队列中的数据就没有必要写入磁盘。因此，我们可通过 SAS 的哈希对象和 PDV 扩展实现数据结构，然后用 SAS 宏封装队列的基本操作。这种实现也可以在 DATA 步内方便调用，实现复杂分析算法。完整的代码如程序 13-3 所示。

程序13-3　SAS宏版的队列实现，存储使用Hash 对象

```
/* 队列 - 初始化：参数为队列名和容量，默认无容量限制；也可指定元素类型 */
%macro QueueDefine(name = Queue1, maxsize =, dataType = n,
```

```
    dataLength = 8, hashexp = 8, rc = );
      length &name._key 8;
      call missing(&name._key);

    %IF &dataType EQ n %THEN %DO;
      length &name._data &datalength;
      retain &name._data 0;
    %END;
    %ELSE %DO;
      length &name._data $ &datalength;
      retain &name._data ' ';
    %END;
    %IF &rc= %THEN %DO;
      %let rc=&name._rc;
    %END;
    declare hash &name._hash(hashexp:&hashexp);
    &rc = &name._hash.defineKey("&name._key");
    &rc = &name._hash.defineData("&name._data");
    &rc = &name._hash.defineDone();

    retain &name._count 0;

    length &name._maxsize 8;
    %IF&maxsize >0 %THEN%DO;
      retain &name._maxsize &maxsize;
    %END;

    retain &name._last 0;
    retain &name._first 1;
%mend;

/* 队列 - 入列  */
%macro QueueEnqueue(name = Queue1,   inputData = , queueLength = , rc = );
  %IF &inputData= %THEN %let inputData=&name._data;
  %IF &queueLength= %THEN %let queueLength=&name._length;
  %IF &rc= %THEN %let rc=&name._rc;

  if &name._maxsize>0 and (&name._count=&name._maxsize) then do;
    put "ERROR:  Queue &name is full!";
    end;
  else do;
    &name._count+1 ;
    &queueLength = &name._count;
    &name._last+1 ;
    &name._key = &name._last;
    &name._data = &inputData;
    &rc = &name._hash.add();
    if &rc ne 0 then put "ERROR:  Internal storage error!"
    &name._last= ;
  end;
%mend;

/* 队列 - 出列*/
%macro QueueDequeue( name = Queue1, OutputData = , queueLength = , rc = );
  %IF &outputData= %THEN %let outputData=&name._data;
  %IF &queueLength= %THEN %let queueLength=&name._length;
  %IF &rc= %THEN %let rc=&name._rc;

  if &name._count <= 0 then do;
    call missing(&outputData);
    &rc = 1;
```

```
    put "ERROR:  Queue &name is empty!";
  end;
  else do;
    &name._key = &name._first;
    &rc = &name._hash.find();
    if &rc ne 0 then put "ERROR:  Internal storage error!"
    &name._last= ;
    &OutputData = &name._data;
    &rc = &name._hash.remove();
    if &rc ne 0 then put "ERROR:  Internal storage error!"
    &name._last= ;
    &name._first+1 ;
    &name._count+(-1);
    &queueLength = &name._count;
  end;
%mend;

/* 队列 - 长度检测  */
%macro QueueLength(name = Queue1, queueLength =  );
  %IF &queueLength= %THEN %let queueLength=&name._length;
  &queueLength = &name._count;
%mend;

/* 队列 - 是否为空 */
%macro QueueIsEmpty(name = Queue1, rc =);
  %IF &rc= %THEN %let rc=&name._rc;
  &rc=(&name._count <= 0);
%mend;

/* 队列 - 清空  */
%macro QueueDelete(name = Queue1,   rc =   );
  %IF &rc= %THEN %let rc=&name._rc;
  &rc = &name._hash.delete();
  if &rc ne 0 then put "ERROR:  Internal storage error!";
%mend;

/* 队列 - 遍历  */
%macro QueueDump(name = Queue1, rc =   );
  %IF &rc= %THEN %let rc=&name._rc;
  if &name._count <= 0 then do;
      put "NOTE:  Queue &name is empty!";
  end;
  else do;
    do id = &name._first to &name._last ;
      &name._key = id;
      &rc = &name._hash.find();
      put "NOTE:  &name[ " id "]="&name._data;
    end;
  end;
%mend;
```

程序 13-4 演示了如何在 DATA 步中使用 SAS 宏版的队列数据结构：

程序13-4　SAS 宏版的队列演示程序

```
data _null_;
  %QueueDefine(name = testQueue, maxsize=5 );
  do i = 1 to 3;
    myvar = i * 10;
    put "NOTE:  入列 " myvar;
    %QueueEnqueue(name = testQueue, inputData = myvar);
  end;
```

```
    %QueueLength(name = testQueue );

    do i = 1 to testQueue_length;
      %QueueDequeue(name = testQueue, OutputData = myvar);
      put "NOTE: 出列 " myvar ;
    end;
    %QueueDelete(name = testQueue, rc = testQueue_rc  );
  run;
```

执行上面的代码后，系统输出与前面FCMP函数版的队列完全相同。由于该实现使用了SAS内置哈希对象，而且运行在内存中不涉及任何磁盘操作，性能要好得多。这种实现方法需要注意的是算法实现不要跟SAS宏中的变量命名发生冲突，因为本质上在SAS的DATA步内变量是没有作用域概念。

13.2 堆栈

堆栈（Stack）在数据结构抽象概念上是一种后进先出（Last-In-First-Out）的线性表（见图13-2），它最基本的两个操作为入栈（Push）和出栈（Pop）。分别将元素压入堆栈，或者将最后加入的元素弹出。有时我们需要Peek操作来访问最后加入的元素，但Peek操作并不从堆栈中弹出最后加入的元素。除了取出元素的顺序不同，堆栈与队列在实现上极其相似，它们都可以通过数组或者链表来实现。实际上，在SAS中实现堆栈只需要对前面的队列实现进行轻微改动即可。

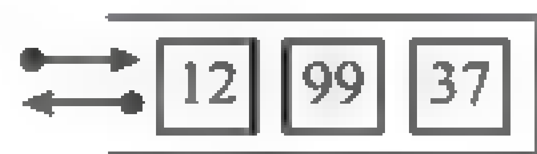


图13-2 堆栈示意图

函数版实现与示例

基于FCMP函数和SAS数据集实现的堆栈见程序13-5所示：

程序13-5 FCMP版的堆栈实现，存储使用持久化数据集

```
proc fcmp outlib=work.funcs.Stack;
  /* 堆栈 - 初始化，参数为名称和容量*/
  function StackDefine(name $, maxsize);
    array y[1] / nosymbols;

    if exist(name) then do;
      array x[1] / nosymbols;
      rc=read_array(name,x);
      length=dim(x);
      x[1]=maxsize;
      rc=write_array(name, x);
    end;
    else do;
```

```
    if maxsize>0 then y[1] maxsize;
    else call missing(y[1]);
    rc write array(name, y);
  end;
  return (rc);
endsub;

/* 堆栈 - 入栈 */
function StackPush(name $, value);
  array y[1] / nosymbols;

  if exist(name) then do;
    array x[1] / nosymbols;
    rc=read_array(name,x);
    length=dim(x);
    maxsize=x[1];
    if maxsize =. | (length-1)< maxsize then do;
      call dynamic_array(y, length+1);
      do i=1 to length;
        y[i]=x[i];
      end;
      y[length+1]=value;
      rc=write_array(name, y);
    end;
    else do;
      put "ERROR:  Stack is full";
      rc=-1;
    end;
  end;
  else do;
    call missing(y[1]);
    call dynamic_array(y, 2);
    y[2]=value;
    rc=write_array(name, y);
  end;
  return (rc);
endsub;

/* 堆栈 - 出栈 */
function StackPop(name $, value);
  outargs value;
  array y[1] / nosymbols;

  if exist(name) then do;
    array x[1] / nosymbols;
    rc=read_array(name,x);
    length=dim(x);
    if length > 1 then do;
      call dynamic_array(y, length-1);
      do i=1 to length-1;
        y[i]=x[i];
      end;
      value=x[length];
    end;
    else do;
      y[1]=x[1];
      put "WARNING:  Stack" name "is EMPTY!";
    end;
    rc write array(name, y);
  end;
  else do;
```

```
    rc=-1;
  end;
  return (rc);
endsub;

/* 堆栈 - 检测长度 */
function StackLength(name $);
  array y[1] / nosymbols;

  if exist(name) then do;
    array x[1] / nosymbols;
    rc=read_array(name,x);
    length=dim(x);
    rc=length-1;
  end;
  else do;
    rc=0;
  end;
  return (rc);
endsub;

/* 堆栈 - 是否为空 */
function StackIsEmpty(name $);
  rc=0;
  if StackLength(name)=0 then rc=1;
  return (rc);
endsub;

/* 堆栈 - 清空 */
function StackDelete(name $);
  array y[1] / nosymbols;

  if exist(name) then do;
    array x[1] / nosymbols;
    rc=read_array(name,x);
    y[1]=x[1];
    rc=write_array(name, y);
  end;
  else do;
    rc=0;
  end;
  return (rc);
endsub;

/* 堆栈 - 遍历  */
function StackDump(name $);
  if exist(name) then do;
    array x[1] / nosymbols;
    rc=read_array(name,x);
    length=dim(x);
    if length > 1 then do;
      do i=1 to length-1;
        value=x[i+1];
        put "NOTE:  " name "[" i "]=" value ;
      end;
    end;
    else do;
      put "WARNING:  Stack" name "is EMPTY!";
    end;
    return(1);
  end;
```

```
    return(0);
  endsub;
run;
quit;
```

程序 13-6 演示了如何在 DATA 步中使用 FCMP 版的堆栈实现。

程序13-6 FCMP版堆栈的应用示例

```
options cmplib=(work.funcs);
data _null_;
  rc=StackDefine('s', 3);

  do i=1 to 3;
    myvar=i*10;
    rc=StackPush('s', myvar);
    put "NOTE: 入栈 " myvar;
  end;

  q_len=StackLength('s');
  do i=1 to q_len;
    rc=StackPop('s', myvar);
    if rc=0 then put "NOTE: 出栈 " myvar ;
  end;

  rc=StackDelete('s');
run;
```

由于堆栈和队列非常相似，可修改队列实现中的元素取出方式来编写宏版本的堆栈。有兴趣的读者可自己完成。实际上，也可将队列和堆栈的实现合二为一，构建一种称为缓冲区 Buffer 的数据类型，分别实现 FIFO 和 LIFO 数据访问方式。堆栈是一种极其有用的数据结构，利用这种数据结构可在 DATA 步内利用 LINK-RETURN 语句构建出可递归调用的函数堆栈机制。

第 14 章

数据结构——链表

14.1 基础知识

链表（Linked List）是一种非顺序存储的线性表，它的每个节点之间通过索引或指针指向下一个相邻节点。与数组不同，链表通常不需要预先知道数据的大小，而是在需要时在链表的任何节点上动态插入和删除。由于每个元素内需要额外的索引或指针域，其空间开销比数组大。链表包括单向链表、双向链表和循环链表等。

数组和链表的应用场景不同，前者注重数据存储密度且按索引访问数组元素，适用于构建没有插入和删除的静态线性表。而链表则适用于数据规模不确定，需要频繁插入 / 删除和动态查找的场景。虽然链表有额外的指针域，但它能提供更加灵活和紧凑高效的数据访问方式。本章介绍如何在 SAS 的程序世界中构建链表这一数据结构。

1. 单向链表

单向链表是由一系列节点组成的线性表，每个节点包含两部分：数据本身以及指向下一节点的指针或引用。单向链表的最后一个节点通常指向一个空值，以此表示链表的结束，如图 14-1 所示。

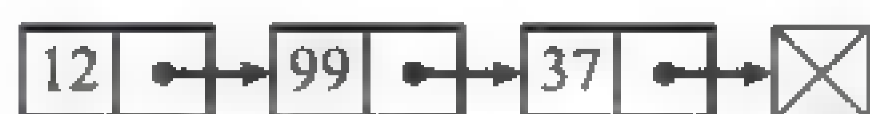


图 14-1　单向链表示意图

单向链表的查找通常由第一个节点（也称为头节点）开始往后遍历，它不支持从某个节点往后回溯。

2. 双向链表

双向链表的每个节点有两个指针 / 引用域，其中一个指向前一个节点，另一个指向后一个节点（见图 14-2）。双向链表可以从任何一个节点开始，向前或向后查找每一个节点。为了表示链表的开始和结束，双向链表的第一个节点的头指针和最后一个节点的尾指针是一个空值，以此表示链表结束。

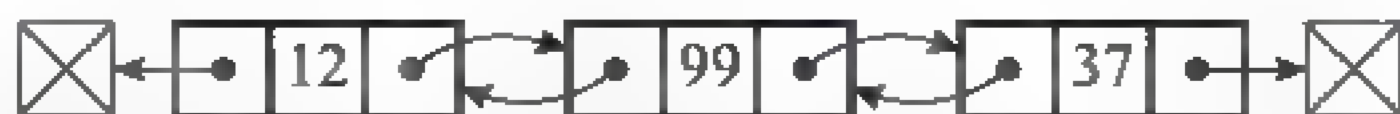


图 14-2　双向链表示意图

3. 循环链表

如果将单向链表的尾指针指向链表的第一个节点，即可构成一个单向循环链表。此时链表可以按照某个方向循环遍历每一个链表元素（见图 14-3）。

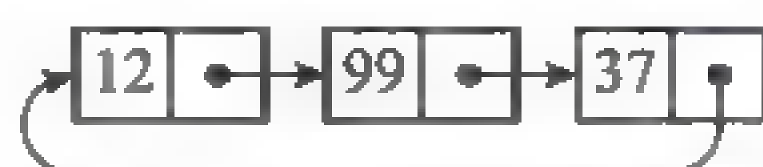


图 14-3　循环链表示意图

如果将双向链表的第一个节点的头指针指向最后一个节点，且将最后一个节点的尾指针指向第一个节点，则双向链表就变成了一个双向循环链表（见图 14-4）。

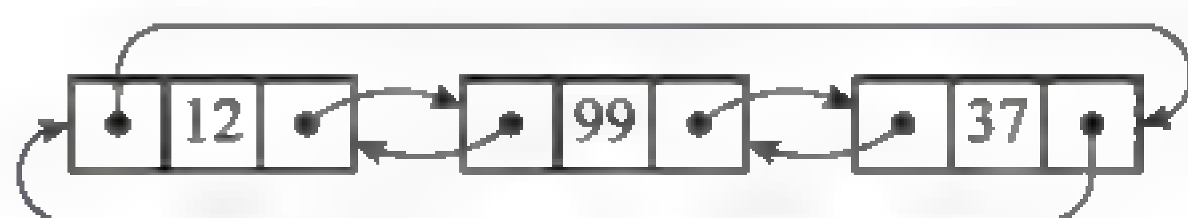


图 14-4　双向循环链表示意图

实际上，我们可以为链表的每个节点设计多个指针，表示不同的查找方向。也可以扩展链表节点的数据部分，让它不再是单个变量，而是一个具有不同长度的顺序表（数组），这样链表就演变为块状链表。由于链表具有数组和指针两大部分，实际上它可以用来构建任何其他的数据结构，如队列、堆栈以及网状邻接表等。这些数据结构的实现要用到指针数据类型，下面我们介绍 SAS 中如何嵌入 C 代码。

14.2　如何在 SAS 代码中内嵌 C 语言代码

SAS 语言的 DATA 步默认并不支持指针数据类型，但 SAS 提供的 PROC PROTO 过程步允许注册并调用 C 语言编写的外部函数，而且函数中可以使用 C 语言数据结构和指针类型。利用 PROC PROTO 过程步注册的 C 语言函数可被 PROC FCMP 和 PROC COMPILE 过程步中定义的函数和例程，以及方法块调用。PROC PROTO 支持在 SAS 代码中直接嵌入 C 语言代码，而不需要额外的 C 语言编译器，SAS 系统内置支持 C 语言代码的编译，这一点为 SAS 语言的扩展以及 SAS 系统需要处理带有指针类型数据结构时提供了无限可能。

为了区分 SAS 语言代码和内嵌的 C 语言代码，此后所有 C 语言代码示例均以斜体表示。比如，我们用 C 语言实现了一个加法计算函数（见程序 14-1），然后我们把它编译输出到 WORK.FUNCS._MATH 包中，实际上它跟 FCMP 包一样是一个 SAS 数据集文件（但其 Owner 列的值为 PROTO）。

程序14-1　PROTO 实现加法函数_ADD

```
proc proto package work.funcs._math;
  double _add(double, double);
  externc _add;
    /*以下为 C 语言实现 _ADD 函数*/
```

```
    double _add(double v1, double v2){
      double sum;
      sum=v1+v2;
      return sum;
    }
    /*以上为 C 语言实现 ADD 函数*/
  externcend;
run;
```

运行上面的 SAS 代码即可将代码编译到 WORK.FUNCS 中，随后就可以在 PROC FCMP 中调用该函数（见程序 14-2）。

程序14-2 FCMP 调用PROTO 中的函数实现_ADD

```
proc fcmp library=work.funcs;
  b=_add(1.5, 2.5);
  put b=;
quit;
```

在 SAS 的输出窗口你可以看到输出结果为 b=4。但这个计算结果是使用内嵌的 C 语言程序计算出来的，并非 SAS 代码计算出来的。既然在 PROC FCMP 可以调用 PROC PROTO 中用 C 语言实现的代码，那么我们是否能够在 DATA 步中调用 PROC PROTO 中用 C 语言编写的函数库呢？如第 11 章“扩展 SAS 功能”所述，我们可使用 PROC FCMP 来进行桥接，直接调用则不行（见程序 14-3）。

程序14-3 将PROTO 函数封装为 FCMP 供 DATA步调用

```
proc fcmp library=work.funcs outlib=work.funcs.math;
  function add(v1, v2);
    s=_add(v1,v2);/*调用 PROTO 中实现的 _ADD 函数*/
    return (s) ;
  endsub;
quit;
```

运行上面的 PROC FCMP 代码即可生成 FCMP 函数 ADD (V1，V2)，就可以在 DATA 步中通过 FCMP 间接调用 PROTO 中用 C 语言实现的函数（见程序 14-4）。

程序14-4 在 DATA步内调用FCMP 函数:DATA->FCMP->PROTO->C

```
options cmplib=work.funcs;
data _null_;
  c=add(1.5, 2.5); /*调用 FCMP 版ADD 函数间接调用 C函数_ADD*/
  put c=;
run;
```

如果 PROTO 输出的函数库 PACKAGE 和 FCMP 输出的函数库 OUTLIB 指定的数据集名称不一致时，DATA 步的调用代码将找不到 PROC PROTO 中定义的 C 语言函数。解决方法是将 PROC PROTO 和 PROC FCMP 输出的函数库设为相同的 SAS 数据集，如都是 WORK.FUNCS；还有一种方法就是在 DATA 步调用时将 PROC PROTO 输出的函数库（如 WORK.CFUNCS）追加到系统选项 CMPLIB 中，这样代码库就能自动找到（如下程序所示）。

```
options cmplib=(work.funcs work.cfuncs);
```

14.3　单向链表和双向链表

有了上面在 SAS 中嵌入 C 语言代码的技术基础，就可以在 SAS 语言中用 C 语言的指针类型来创建单向链表数据结构。首先要用标准的 C 语言风格创建结构体 struct LinkList（如以下程序所示）：

```
struct LinkList{
  double value; /*数据部分*/
  struct LinkList * next; /*结构指针*/
};
```

然后基于该结构体用 C 语言实现链表的基本操作，主要包括创建 (CREATE)、追加 (APPEND)、插入(PREPEND)、尾部追加(ADDNODE)、删除(DELETE)和查找(FIND) 等基本操作。整个链表的完整定义如程序 14-5 所示，PROC PROTO 过程步输出的函数包名 WORK.FUNCS.LINKLIST，表示函数编译的代码放在 WORK.FUNCS 数据集中，其中第 3 级 LINKLIST 只是数据集下一级的模块表示。

程序14-5　单向链表的完整实现

```
proc proto package=work.funcs.linklist;
  /*用 C 语言定义链表节点*/
  struct LinkList{
    double value;/*数据部分*/
    struct LinkList * next; /*结构指针*/
  };

  /*用 C 语言创建链表，返回新建节点的指针*/
  struct LinkList * create(double);
  externc create;
    struct LinkList * create(double value){
      struct LinkList * element = (struct LinkList  *)
        malloc( sizeof( struct LinkList));
      element->value=value;
      element->next=0;
      return element;
    }
  externcend;

  /*后面追加-返回新建节点指针*/
  struct LinkList * append(struct LinkList * node, double value);
  externc append;
    struct LinkList * append(struct LinkList * node, double value){
      struct LinkList * element  = (struct LinkList  *)
        malloc( sizeof( struct LinkList));
      element->value=value;
      element->next=0;
      if (node != 0)
        element->next=node->next;
      node->next= element;
      return element;
    }
  externcend;

  /*前面插入-返回新建节点指针*/
  struct LinkList * prepend(struct LinkList * node, double value);
```

```
externc prepend;
  struct LinkList * prepend(struct LinkList * node, double value){
    struct LinkList * element  = (struct LinkList  *)
      malloc( sizeof( struct LinkList));
    element->value-value;
    element->next=0;
    if (node != 0)
      element->next=node;
    return element;
  }
externcend;

/*尾部追加-返回新建节点指针*/
struct LinkList * addnode(struct LinkList * node, double value);
externc addnode;
  struct LinkList * addnode(struct LinkList * node, double value){
    struct LinkList * lastnode  = (struct LinkList  *)
      malloc( sizeof( struct LinkList));

    struct LinkList * element  = (struct LinkList  *)
      malloc( sizeof( struct LinkList));
    element->value=value;
    element->next=0;

    while (node != 0)
    {
      lastnode=node;
      node=node->next;
    }
    lastnode->next= element;
    return element;
  }
externcend;

/*删除节点-返回下一节点指针*/
struct LinkList * delete(struct LinkList * node, double value);
externc delete;
  struct LinkList * delete(struct LinkList * node, double value){
    struct LinkList * lastnode  = (struct LinkList  *)
      malloc( sizeof( struct LinkList));
    while (node != 0)
    {
      if (node->value==value )
      {
        lastnode->next=node->next;
        node->next=0;/*隔离*/
        return lastnode->next;
      }
      lastnode=node;
      node=node->next;
    }
    return 0;
  }
externcend;

/*查找节点-返回匹配节点指针*/
struct LinkList * find(struct LinkList * node, double value);
externc find;
  struct LinkList * find(struct LinkList * node, double value){
    while (node !  0)
    {
```

```
      if (node->value == value )
        return node;
      node=node->next;
    }
    return 0;
  }
externcend;
run;
```

在 SAS 代码中如何使用该链表结构呢？程序 14-6 创建了一个值为 12 的头节点并添加了值为 99 和 37 的节点。然后我们遍历链表，并在头节点处插入值为 88 的节点，随后进行查找和删除等操作。在 PROC FCMP 中可使用 C 语言定义的结构体，但 DATA 步中却不能支持 PROC PROTO 中定义的 C 语言结构体，即使如此，依然可用 FCMP 桥接两者。

程序14-6　单向链表的应用示例（FCMP）

```
proc fcmp library=work.funcs;
  struct LinkList node;
  node=create(12);

  struct LinkList header;
  header=node; /*保存第一个节点*/

  node=append (node, 99);
  node=append (node,37);

  put "遍历";
  node=header;
  put node.value ;
  do while(^isnull(node.next));
    node=node.next;
    put node.value;
  end;

  put '  ';
  put "插入 88";
  header=prepend(header,88);

  node=header;
  put node.value ;
  do while(^isnull(node.next));
    node=node.next;
    put node.value;
  end;

  put '  ';
  put "查找 99 ";
  node=find(header,99);
  if ^missing(node.value) then do;
      put node.value;
  end;
  else do;
      put "N/A";
  end;

  put '  ';
  put "查找 39 ";
  node=find(header,39);
```

```
      if ^missing(node.value) then do;
          put node.value;
      end;
      else do;
          put "N/A";
      end;

      put '  ';
      put "删除 12";
      node=delete(header,12);

      node=header;
      put node.value ;
      do while(^isnull(node.next));
        node=node.next;
        put node.value;
      end;
    run;
  quit;
```

在 SAS 输出窗口中可以看到系统输出如图 14-5 所示。

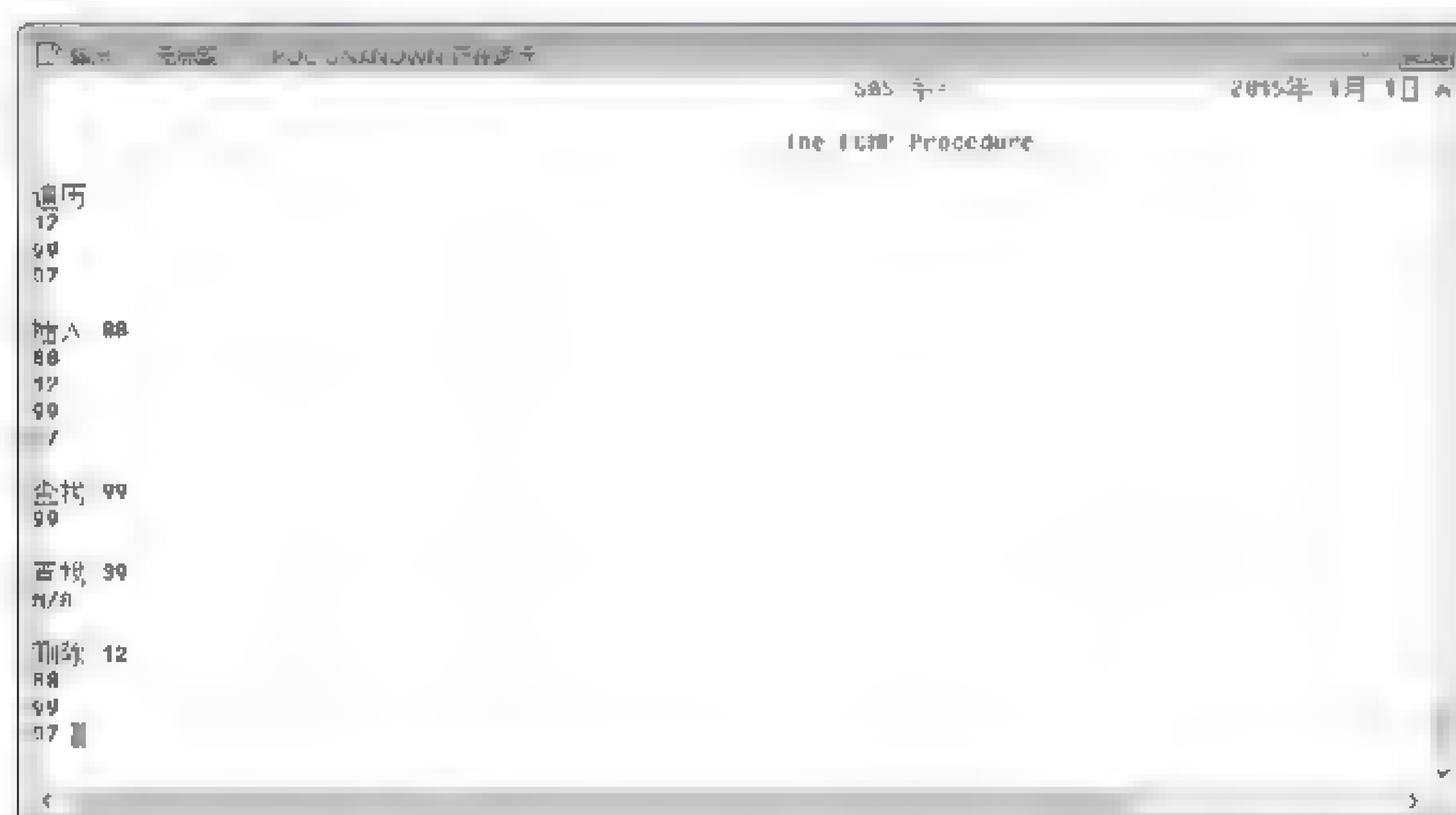


图 14-5　FCMP 链表测试程序输出

同理，我们可以实现双向链表 DualLinkList 如程序 14-7 所示。双向链表的基本操作与单向链表基本相同，但追加、插入、删除和查找会涉及头尾指针的修改问题。双向链表的删除和查找可以按照向前或者向后两种方向进行。

程序14-7　双向链表的完整实现

```
proc proto package=work.funcs.duallinklist;
  struct DualLinkList{
    double value;              /*数据部分*/
    struct DualLinkList * next; /*结构指针*/
    struct DualLinkList * prev; /*结构指针*/
  };

/*创建节点*/
struct DualLinkList * create(double);
externc create;
  struct DualLinkList * create(double value){
    struct DualLinkList * element = (struct DualLinkList  *)
      malloc( sizeof( struct DualLinkList));
    element->value=value;
```

```
        element->next=0;
        element->prev=0;/*前一个节点*/
        return element;
    }
externcend;

/*链表当前位置插入，其后一个节点成为插入节点的下一个节点*/
struct DualLinkList * append(struct DualLinkList * node, double
    value);
externc append;
    struct DualLinkList * append(struct DualLinkList * node,
        double value){
        struct DualLinkList * element  = (struct DualLinkList  *)
          malloc( sizeof( struct DualLinkList));
        element->value=value;
        element->prev=node;
        if (node != 0)
        {
          element->next=node->next;
          if (node->next !=0 )  (node->next)->prev=element;

          node->next=element;
        }
        return element;
    }
externcend;

/*前面插入-返回新建节点*/
struct DualLinkList * prepend(struct DualLinkList * node, double
    value);
externc prepend;
    struct DualLinkList * prepend(struct DualLinkList * node, double
        value){
        struct DualLinkList * element  = (struct DualLinkList  *)
          malloc( sizeof( struct DualLinkList));
        element->value=value;
        element->next=node;
        if (node !=0)
        {
          element->prev=node->prev;
          if (node->prev !=0)  (node->prev)->next=element;

          node->prev=element;
        }
        return element;
    }
externcend;

/*链表的尾部追加/头部插入*/
struct DualLinkList * addnode(struct DualLinkList * node, double value,
    int direction);
externc addnode;
    struct DualLinkList * addnode(struct DualLinkList * node,
        double value, int direction){
        struct DualLinkList * element  = (struct DualLinkList  *)
          malloc( sizeof( struct DualLinkList));
        struct DualLinkList * lastnode  = (struct DualLinkList  *)
          malloc( sizeof( struct DualLinkList));

        element->value=value;
        element->next=0;
```

```
      while (node != 0)
      {
        lastnode=node;
        if (direction==0) node=node->next;
        else              node=node->prev;
      }

      if (direction== 0) {
        lastnode->next= element;
        element->prev=lastnode;
      }
      else {
        lastnode->prev= element;
        element->next=lastnode;
      }
      return element;
    }
  externcend;

  /*删除节点-返回下一节点*/
  struct DualLinkList * delete(struct DualLinkList * node, double
    value, int direction);
  externc delete;
    struct DualLinkList * delete(struct DualLinkList * node,
      double value, int direction){
      struct DualLinkList * lastnode  = (struct DualLinkList  *)
        malloc( sizeof( struct DualLinkList));
      while (node != 0)
      {
        if (node->value==value )
        {
          lastnode->next=node->next;
          if (node->next !=0)  (node->next)->prev=lastnode;
          node->next=0;
          node->prev=0;
          return lastnode->next;
        }
        lastnode=node;
        if (direction==0) node=node->next;
        else              node=node->prev;
      }
      return 0;
    }
  externcend;

  /*链表查找*/
  struct DualLinkList * find(struct DualLinkList * node, double
    value, int direction);
  externc find;
    struct DualLinkList * find(struct DualLinkList * node, double
      value, int direction){
      while (node != 0)
      {
        if (node->value==value )
          return node;
        if (direction==0) node=node->next;
        else              node=node->prev;
      }
      return 0;
    }
  externcend;
run;
```

程序14-8演示了双向链表基本应用，读者可以将它进一步封装在DATA步内进行调用。

程序14-8 双向链表的应用示例（FCMP）

```
proc fcmp libname=work.funcs;
  struct DualLinkList node;
  node=create(12);

  struct DualLinkList header;
  header=node;/*保存第一个节点*/

  node=append(node, 99);
  node=append(node,37);

  struct DualLinkList tail;
  tail=node;/*保存最后一个节点*/

  put "正向遍历";
  node=header;
  put node.value ;
  do while(^isnull(node.next));
    node=node.next;
    put node.value ;
  end;

  put "后向遍历";
  node=tail;
  put node.value ;
  do while(^isnull(node.prev));
    node=node.prev;
    put node.value ;
  end;

  put '  ';
  put "插入 88";
  header=prepend(header,88);

  node=header;
  put node.value ;
  do while(^isnull(node.next));
    node=node.next;
    put node.value ;
  end;
  put '  ';
  put "查找 99 ";
  node=find(header,99, 0);
  if ^missing(node.value) then do;
      put node.value ;
  end;
  else do;
      put "N/A";
  end;

  put '  ';
  put "查找 39 ";
  node=find(header,39, 0);
  if ^missing(node.value) then do;
      put node.value ;
  end;
  else do;
      put "N/A";
  end;
```

```
    put '  ';
      put "删除 12";
      node=delete(header,12,0);

      node=header;
      put node.value ;
      do while(^isnull(node.next));
        node=node.next;
        put node.value;
      end;
    run;
quit;
```

系统输出如图 14-6 所示。

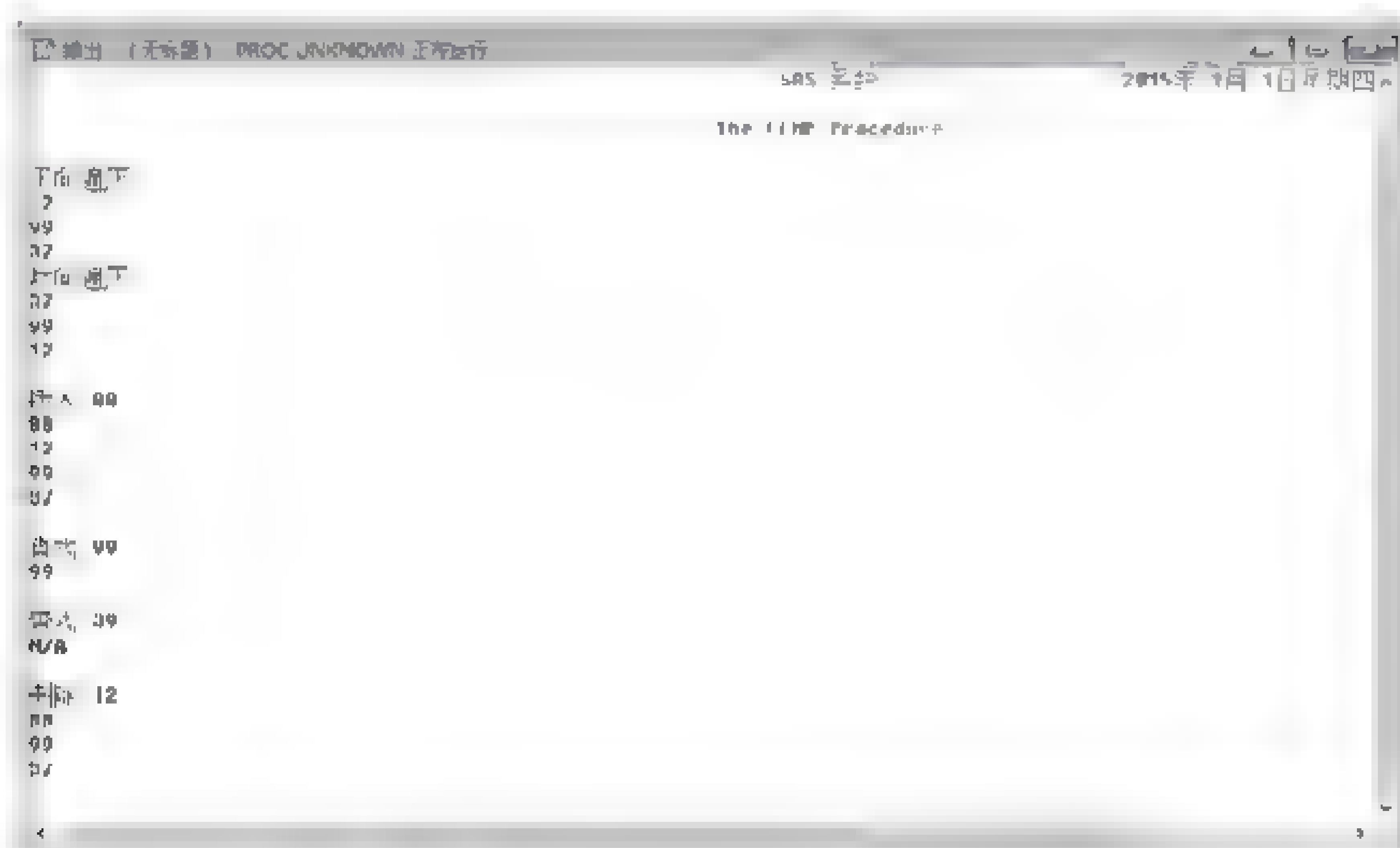


图 14-6　FCMP 双向链表测试程序输出

14.4　链表应用：约瑟夫斯问题

为了演示链表的应用，我们用 SAS 求解约瑟夫斯问题。据说公元 1 世纪有个犹太人叫提图斯・弗拉维奥・约瑟夫斯 (Tifus Flavins Josephus)，他和 40 个战友在罗马人占领塞约塔巴后躲到了一个山洞里。于是这群人在被长期围困时讨论了接下来可能的出路：自杀还是投降？最后集体的决定就是自杀。然而约瑟夫斯和另外一个同伴却并不赞成自杀计划，于是狡猾的他建议采用如下方式逐个自杀：41 个人站成一个圆圈，由第 1 个人开始报数，每数到 3 的人执行自杀计划，活着的人监督执行情况；自杀后由下一个人重新从 1 开始报数，依次自杀直到所有的人都自杀完毕为止。于是大家开始排队，而约瑟夫斯和另一个同伴心领神会地站到了某两个特别的位置上，一起神奇地逃过了这场死亡游戏，最后活着向罗马人投降了！

这个问题有个现代版的游戏就是 N 个玩家围成一圈，游戏规定从第 1 个玩家开始报数，凡是报数为 M 的玩家将被淘汰出局；下一个玩家继续从 1 开始报数，报数为 M 的玩家依次出列直到剩下最后一个人为止即胜出。这个游戏归结为一般性数学问题就是：给定的 N 和 M，求解出局顺序或最终胜出玩家的位置编号 X。约瑟夫斯问题就是该游戏

在 N 41 和 M 3 时求解最后两位玩家的位置问题。

这个问题可用单向循环链表求解，首先需实现循环链表 CircularLinkList 数据结构。完整的代码如下所示（见程序 14-9）：

程序14-9　单向循环链表的完整实现

```
proc proto package=work.funcs.circularLinkList;
  struct CircularLinkList{
    double value;                /*数据部分*/
    struct CircularLinkList * next; /*结构指针*/
  };

  /*创建节点*/
  struct CircularLinkList * create(double);
  externc create;
    struct CircularLinkList * create(double value){
      int i;
      struct CircularLinkList * element = (struct CircularLinkList  *)
        malloc( sizeof( struct CircularLinkList));
      element->value=value;
      element->next=element;
      return element;
    }
  externcend;

  /*追加节点：创建节点并追加到当前节点后面*/
  struct CircularLinkList * append(struct CircularLinkList * node,
    double value);
  externc append;
    struct CircularLinkList * append(struct CircularLinkList * node,
      double value){
      struct CircularLinkList * element  = (struct CircularLinkList *)
        malloc( sizeof( struct CircularLinkList));
      element->value=value;
      element->next=element;
      if (node != 0)
      {
        element->next=node->next;
        node->next=element;
      }
      return element;
    }
  externcend;

  /*插入节点：创建节点并插入到当前节点前面*/
  struct CircularLinkList * prepend(struct CircularLinkList * node,
    double value);
  externc prepend;
    struct CircularLinkList * prepend(struct CircularLinkList * node,
      double value){
      struct CircularLinkList * element   = (struct CircularLinkList *)
        malloc( sizeof( struct CircularLinkList));
      struct CircularLinkList * lastnode  = node;
      element->value=value;
      element->next =node;

      while (node != 0 )
      {
        lastnode=node;
        node=node->next;
```

```
      if (node == lastnode) break;
    }
    if (lastnode != 0) lastnode->next= element;

    return element;
  }
externcend;
/*删除节点：找到匹配值的节点并删除之*/
struct CircularLinkList * delete(struct CircularLinkList * node,
  double value);
externc delete;
  struct CircularLinkList * delete(struct CircularLinkList * node,
    double value){
    struct CircularLinkList * element;
    struct CircularLinkList * orignode  = node;
    struct CircularLinkList * lastnode  = 0;

    struct CircularLinkList * tempnode  = node;

    while (node != 0)
    {
      if (node->value==value )
      {
        if (lastnode==0) {
          tempnode=node;
          while(tempnode != 0)
          {
            lastnode=tempnode;
            tempnode=tempnode->next;
            if (tempnode == orignode) break;
          }
        }
        lastnode->next=node->next;
        node->next=node;
        return lastnode->next;
      }
      lastnode=node;
      node=node->next;
      if (node == orignode) break;
    }
    return 0;
  }
externcend;
/*查找节点：找到匹配值的节点*/
struct CircularLinkList * find(struct CircularLinkList * node,
  double value);
  externc find;
  struct CircularLinkList * find(struct CircularLinkList * node,
    double value){
    struct CircularLinkList * element;
    struct CircularLinkList * orignode  = node;

    while (node != 0)
    {
      if (node->value==value )
        return node;
      node=node->next;
      if (node == orignode) break;
    }
    return 0;
  }
```

```
    externcend;
    /*判断两个节点是否为同一节点*/
    int equal(struct CircularLinkList * node, struct CircularLinkList *
      node2 );
    externc equal;
      int equal(struct CircularLinkList * node, struct CircularLinkList
        * node2){
        return node==node2;
      }
    externcend;
    /*删除节点：删除当前节点并返回下一个节点*/
    struct CircularLinkList * deleteNode(struct CircularLinkList * node );
    externc deleteNode;
      struct CircularLinkList * deleteNode(struct CircularLinkList * node){
        struct CircularLinkList * element;
        struct CircularLinkList * orignode  = node;
        struct CircularLinkList * lastnode  = 0;

        while (node != 0)
        {
          lastnode=node;
          node=node->next;
          if (node == orignode) break;
        }
        lastnode->next=node->next;
        node->next=node;
        return lastnode->next;
      }
    externcend;
  run;
  quit;
```

为了能在 DATA 步中使用单向循环链表，我们还需要建立一个 FCMP 函数 Josephus 来桥接 PROTO 代码进行求解（见程序 14-10），其中 n 和 m 为输入参数，分别表示玩家数 N 和玩家出局报数 M，而数组 r[] 为用来存放出局顺序的输出参数。注意，由于 DATA 步不支持结构体，而 FCMP 支持结构体，因此我们通常需要将结构体的概念封装在 FCMP 函数内，而 DATA 步一般用线性数组传递数据给 FCMP 进行调用。

程序14-10 FCMP 封装单向循环链表结构体桥接DATA 步和PROTO 函数

```
proc fcmp libname=work.funcs outlib=work.funcs.josephus;
  function Josephus(n, m , r[*]);
    outargs r;
    struct CircularLinkList node;
    struct CircularLinkList header;

    node=create(1);/*创建N个元素的单向循环链表*/
    header=node;
    do i=2 to n;
      node=append(node, i);
    end;

    node=header;
    i=1;
    c=1;
    do while(^isnull(node.next));
      node=node.next;
      i=i+1;
      if i=m then do;
```

```
        r[c]=node.value; /*保存出局玩家编号*/
        c=c+1;
        node=deleteNode(node);/*玩家出局*/
        i=1;
      end;
      if equal(node, node.next) then do;
        r[c]=node.value;
        return(1);/*游戏结束*/
      end;
    end;
    return(1);
  endsub;
  run;
quit;
```

最后，在 DATA 步中调用 FCMP 函数 Josephus 来实现求解 (见程序 14-11)，只需要传入参数 n 和 m，以及用于接收计算结果的一维数组 seq 即可。

程序14-11 单向循环链表求解约瑟夫斯问题的SAS 演示程序

```
options cmplib=(work.funcs);
data _null_;
  array seq [41];
  n=41; m=3;
  r=Josephus(n, m, seq);
  put "Josephus (" n= m= ")";
  put seq[*];
run;
```

运行上面的代码后系统输出如下：

```
Josephus (n=41 m=3 )
3 6 9 12 15 18 21 24 27 30 33 36 39 1 5 10 14 19 23 28 32 37 41 7 13 20 26 34
40 8 17 29 38 11 25 2 22 4 35 16 31
```

该输出序列表示 41 个士兵编号的自杀顺序，因此如果 Jossephus 和另一个同伴想要活命的话，在排队的时候就应该尽可能抢占死亡轮盘上的最后两个位置：第 16 个和第 31 个位置（见图 14-7）。如果有 3 个人想要从该游戏中存活下来，则应该分别占据 35、16 和 31 这三个位置。



图 14-7 约瑟夫斯死亡轮盘

第 15 章

数据结构——二叉树

二叉树（Binary Tree, BTree）是每个节点最多有两个子节点构成的树状数据结构，左右子节点及其子集构成的集合通常被称为左子树和右子树（见图 15-1）。

假定二叉树的层次为 k，则二叉树最多可有 $n=2^k-1$ 个节点。比如 $k=4$，则最多可有 $n=2^4-1=15$ 个节点。同理，n 个节点可构成深度为 $k=\log2(n+1)$ 的完全二叉树。

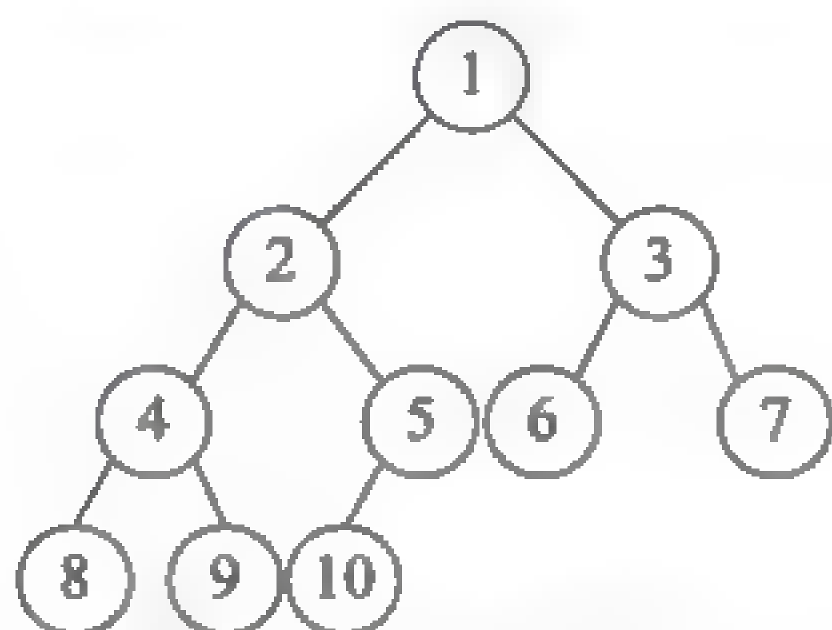


图 15-1　二叉树范例

如果以顺序序列方式存储由 n 个节点构成的二叉树，则对于编号为 i 的节点，其父节点的编号为 $i/2$ 取整。其中，如果 $i \leqslant n/2$ ，则其左侧子节点编号为 $2i$，否则该节点没有左侧子节点；如果 $i \leqslant (n-1)/2$ ，其右侧子节点编号为 $2i+1$，否则该节点没有右侧子节点。比如图 15-1 中 $n=10$，编号 $i=4 \leqslant 10/2$ ，则其左侧子节点编号为 $2i=8$，且 $i=4 \leqslant (10-1)/2$ 则其右侧子节点编号为 $2i+1=9$。

树状数据结构蕴含了彼此之间的聚合关系，它最重要的操作是遍历每个节点。遍历二叉树的顺序包括深度优先和广度优先两大类，其中深度优先包括前序遍历、中序遍历和后序遍历 3 种方式，其遍历方法可递归定义如下。

- 深度优先

（1）前序遍历：访问根节点，【前序遍历】左子树，【前序遍历】右子树。

（2）中序遍历：【中序遍历】左子树，访问根节点，【中序遍历】右子树。

（3）后序遍历：【后序遍历】左子树，【后序遍历】右子树，访问根节点。

- 广度优先

按照层次逐级访问整棵树：先访问根节点，然后是左侧子节点，最后是右侧子节点。

15.1　PROTO 实现与封装

在 SAS 中实现二叉树有多种方式，有了前面 PROC PROTO 创建指针链表的经验，

在 SAS 中实现二叉树就非常容易。首先是要建立二叉树节点的结构体 BTree，其定义如程序 15-1 所示。

程序15-1　BTree 结构体

```
struct BTree{
  double value;/*数据部分*/
  struct BTree * leftChild;  /*指向左节点的指针*/
  struct BTree * rightChild; /*指向右节点的指针*/
};
```

然后就需要创建二叉树节点的函数 create（见程序 15-2），假设节点中数据部分为一个双精度浮点数 double 值，则需要把它作为参数传递给节点创建函数。

程序15-2　创建一个BTree节点

```
struct BTree * create(double value){
  int i;
  struct BTree * node=0;
  node = (struct BTree  *) malloc( sizeof( struct BTree));
  node->value=value;
  node->leftChild=0;
  node->rightChild=0;
  return node;
}
```

为了将子节点和父节点链接起来，需要将节点关联起来形成“树”的函数 setChild（见程序 15-3），其参数为父节点和子节点的指针，以及指示将子节点当作左节点还是右节点的标志量，right=1 表示当作右节点，否则默认为左节点。

程序15-3　设置BTree节点关系的函数setChild

```
struct BTree * setChild(struct BTree * parent, struct BTree * child, int right){
  if ( parent!=0){
    if (right!=1) parent->leftChild=child;
    else parent->rightChild=child;
  }
  return parent;
}
```

由于 SAS 语言中最方便和常见的数据存储是线性数组，因此我们设计 build 函数（见程序 15-4）实现从线性数组 value[] 中构建一个完全二叉树的方法。

程序15-4　从线性数组构建完全二叉树

```
struct BTree * build(struct BTree * p, double * value, int length, int pos ){
  struct BTree * node;
  p->value= value[pos] ;
  if ( ( value[ pos *2 +1 ] )==0 || (pos *2 +1)>(length-1) )
    p->leftChild=0;
  else
  {
    node= (struct BTree  *) malloc( sizeof( struct BTree));
    p->leftChild= node;
    build( p-> leftChild, value, length, pos *2 +1);
  }

  if ( ( value[ pos *2 +2 ]) ==0 || (pos *2 +2) > length-1)
    p->rightChild=0;
  else
```

```
  {
    node (struct BTree  *) malloc( sizeof( struct BTree));
    p->rightChild= node;;
    build( p-> rightChild, value, length, pos *2 +2);
  }

  return p;
}
```

至此，建立二叉树数据结构的函数就基本准备完毕。后面可根据二叉树遍历方法的定义实现各种二叉树遍历函数。注意函数实现中根据定义采用了递归调用。

1. 前序遍历算法实现（见程序15-5）

程序15-5　二叉树的前序遍历算法实现

```
struct BTree * iterativePreOrder(struct BTree * root, double * value, int * pos){
  value[*pos] = root->value;
  *pos=*pos+1;
  if (root->leftChild  !=0)
    iterativePreOrder( root->leftChild, value, pos);
  if (root->rightChild !=0)
    iterativePreOrder( root->rightChild, value, pos);
  return root;
}
```

2. 中序遍历算法实现（见程序15-6）

程序15-6　二叉树的中序遍历算法实现

```
struct BTree * iterativeInOrder(struct BTree * root, double * value, int * pos){
  if (root->leftChild  !=0)
    iterativeInOrder( root->leftChild, value,  pos);

  value[*pos] = root->value;
  *pos= *pos +1;

  if (root->rightChild !=0)
    iterativeInOrder( root->rightChild, value, pos);
  return root;
}
```

3. 后序遍历算法实现（见程序 15-7）

程序15-7　二叉树的后序遍历算法实现

```
struct BTree * iterativePostOrder(struct BTree * root, double * value, int* pos){
  if (root->leftChild  !=0)
    iterativePostOrder( root->leftChild, value,  pos);
  if (root->rightChild !=0)
    iterativePostOrder( root->rightChild, value,  pos);

  value[*pos] = root->value;
  *pos=*pos+1;
  return root;
}
```

4. 广度优先遍历算法实现（见程序15-8）

程序15-8 二叉树的广度优先遍历算法实现

```
struct BTree * iterativeLayerOrder(struct BTree * root, double * value, int * pos){
  int head=0, tail=0;
  struct BTree * node;
  struct BTree * p[100];

  if (root ==0) return root;
  p[head]=root;
  tail++;

  while(head< tail)
  {
    node=p[head];
    if(node->leftChild != 0)
    {
      p[tail] = node->leftChild;
      tail++;
    }
    if(node->rightChild != 0)
    {
      p[tail] = node->rightChild;
      tail++;
    }

    value[*pos] = p[head]->value;
    *pos=*pos+1;

    head++;
  };
  return root;
}
```

为了让上面C语言实现的二叉树代码能够在SAS中运行，要用PROC PROTO过程步将所有这些C函数进行封装。也就是说需要把上面的代码放到如下对应注释之后（见程序15-9），然后在SAS中运行即可。SAS自带C语言编译器处理嵌入的C语言代码。这跟一些C++编译器能够处理C代码，以及C编译器能够处理内嵌的宏汇编代码机制是一样的。另外程序15-9还定义了一个判断树是否为空的函数isTreeNull。

程序15-9 二叉树完整实现代码

```
proc proto package=work.funcs.btree;
  /*结构体 C代码放在此处*/

  struct BTree * create(double);
  externc create;
    /*创建根节点的 C代码放在此处*/
  externcend;

  struct BTree * setChild(struct BTree *, struct BTree *, int right);
  externc setChild;
    /*设置子节点的 C代码放在此处*/
  externcend;

  struct BTree * build(struct BTree *, double *, int, int );
  externc build;
    /*从线性存储创建二叉树的 C 代码放在此处*/
```

```
    externcend;

    struct BTree * iterativePreOrder(struct BTree *, double *, int *);
    externc iterativePreOrder;
      /*前序遍历 C 代码放在此处*/
    externcend;

    struct BTree * iterativeInOrder(struct BTree *, double *, int *);
    externc iterativeInOrder;
      /*中序遍历 C 代码放在此处*/
    externcend;

    struct BTree * iterativePostOrder(struct BTree *, double *, int *);
    externc iterativePostOrder;
      /*后序遍历 C 代码放在此处*/
    externcend;

    struct BTree * iterativeLayerOrder(struct BTree *, double *, int *);
    externc iterativeLayerOrder;
      /*层次遍历 C 代码放在此处*/
    externcend;

    int isTreeNull(struct BTree * tree);
    externc isTreeNull;
      int isTreeNull(struct BTree * tree){
        return tree==0;
      }
    externcend;
run;
```

至此，就可以在 PROC FCMP 中测试所创建的二叉树结构以及各种遍历算法。如程序 15-10 所示以图 15-1 中的二叉树为例进行说明。

程序15-10　二叉树功能的简单测试程序

```
proc fcmp libname=work.funcs;
    array a[10] (1 2 3 4 5 6 7 8 9 10);

    struct BTree root;
    root=build(root, a, dim(a), 0);

    array b[10];
    put "前序遍历";
    p=0;
    root=iterativePreOrder(root, b, p);
    put b= ;

    put "中序遍历";
    p=0;
    root=iterativeInOrder(root, b, p);
    put b=;

    put "后序遍历";
    p=0;
    root=iterativePostOrder(root, b, p);
    put b= ;

    put "广度遍历";
    p=0;
    root=iterativeLayerOrder(root, b, p);
    put b=;
```

```
run;
quit;
```

运行上面的SAS代码后，系统在SAS输出窗口程序输出如下结果：

```
前序遍历： 1 2 4 8  9  5 10 3 6 7
中序遍历： 8 4 9 2  10 5 1  6 3 7
后序遍历： 8 9 4 10 5  2 6  7 3 1
广度遍历： 1 2 3 4  5  6 7  8 9 10
```

由于SAS语言的DATA步不支持结构和指针处理，因此只能在PROC FCMP中可使用结构体数据。如果我们需要在DATA步中使用二叉树遍历，则只能通过线性数组传入传出FCMP子系统进行处理。为此设计FCMP函数iterativeBTree来封装整个过程，它能够对给定SAS数组建立完全二叉树，然后按照指定的4种二叉树遍历顺序输出结果到另一个SAS数组中。其完整的代码如程序15-11所示。

程序15-11 BTree 实现的FCMP 封装

```
proc fcmp inlib=work.funcs outlib=work.funcs.struct;
  function iterativeBTree(a [*] , b[*], mode);
    outargs b;

    struct BTree root;
    root=build(root, a, dim(a), 0); /*从输入数组中创建二叉树*/

    p=0;
    if mode=1 then
      root=iterativePreOrder(root, b, p);
    else if mode=2 then
      root=iterativeInOrder(root, b, p);
    else if mode=3 then
      root=iterativePostOrder(root, b, p);
    else
      root=iterativeLayerOrder(root, b, p);
    return (1);
  endsub;
run;
quit;
```

上面的FCMP函数编译通过后，就可以使用程序15-12所示的DATA步进行测试。

程序15-12 BTree 的简单DATA 步测试程序

```
options cmplib=work.funcs;
data _null_;
  array a[10] _temporary_ (1 2 3 4 5 6 7 8 9 10);
  array b[10] _temporary_;

  do mode=1 to 4;/*四种模式进行遍历*/
    rc=iterativeBTree(a,b,mode);
    do i=1 to dim(b);
      put b[i] @@;
    end;
    put "<-" mode ;
  end;
run;
quit;
```

运行上面的代码后，SAS在日志窗口中将输出如下结果：

```
1 2 4 8 9 5 10 3 6 7 <-1
8 4 9 2 10 5 1 6 3 7 <-2
8 9 4 10 5 2 6 7 3 1 <-3
1 2 3 4 5 6 7 8 9 10 <-4
```

15.2　FCMP 二叉树实现

不用 PROC PROTO 内嵌 C 代码也可基于线性数组创建二叉树，程序 15-13 是利用 FCMP 函数和 SAS 数据集模拟实现的二叉树机制。不过由于此时用到了涉及 I/O 的外部临时数据集，本方法仅适用于性能要求不那么高的场合。

程序15-13　BTree 的FCMP 版实现

```
proc fcmp outlib=work.funcs.btree2;
  /*定义二叉树*/
  function BTreeDefine(name $, maxsize);
    array y[1] / nosymbols;

    if exist(name) then do;
      array x[1] / nosymbols;
      rc=read_array(name,x);
      length=dim(x);
      x[1]=maxsize;
      rc=write_array(name, x);
      put "WARNING: Change BTree" name "(maxsize=" maxsize ")";
    end;
    else do;
      if maxsize>0 then y[1]=maxsize;
      else call missing(y[1]);
      rc=write_array(name, y);
      put "NOTE: Create BTree" name "(maxsize=" maxsize ")";
    end;
    return (rc);
  endsub;

  /*从数组中创建二叉树*/
  function BTreeBuild(name $, value[*]);
    array y[1] / nosymbols;

    inputLength=dim(value);
    minlen=inputLength;
    maxsize=.;
    if exist(name) then do;
      array x[1] / nosymbols;
      rc=read_array(name,x);
      length=dim(x);
      maxsize=x[1];
      if maxsize >0 then minlen=min(maxsize, inputLength);
    end;

    call dynamic_array(y, minlen+1);
    y[1]=maxsize;
    do i=1 to minlen;
      y[i+1]=value[i];
    end;
    rc=write_array(name, y);
```

```
  put "NOTE:Build" value ">> BTree" name;
  return (rc);
endsub;

/*返回根节点 ID，永远为 1 */
function BTreeGetRootId(name $);
  return (1);
endsub;

/*返回 id 节点的子节点： right=0 左节点 right=1 右节点*/
function BTreeGetChildId(name $, id, right);
  if (right ^=1) then newid= 2 * (id-1) +1;
  else                newid= 2 * (id-1) +2;
  if exist(name) then do;
    array x[1] / nosymbols;
    rc=read_array(name,x);
    length=dim(x);
    maxsize=x[1];
    if (newid+1< length) then return (newid+1);
  end;
  return (.);
endsub;
/*返回 id 节点的值*/
function BTreeGetNodeValue(name $, id);
  if exist(name) then do;
    array x[1] / nosymbols;
    rc=read_array(name,x);
    length=dim(x);
    maxsize=x[1];
    if 1<= id & id<= length then return( x[id+1] );
  end;
  return (.);
endsub;

/*前序遍历*/
function iterativePreOrder(name $, id, value[*], pos);
  outargs value;
  outargs pos;
  if exist(name) then do;
    array x[1] / nosymbols;
    rc=read_array(name,x);
    length=dim(x);
    maxsize=x[1];

    if 0< id & id<= length then do;
      value[pos]=x[id+1];
      pos=pos+1;
    end;

    leftChildId=BTreeGetChildId(name, id, 0);
    if leftChildID ^=. then rc=iterativePreOrder(name , leftChildId,
      value, pos);

    rightChildId BTreeGetChildId(name, id, 1);
    if rightChildID ^=. then rc=iterativePreOrder(name ,
      rightChildId, value, pos);
  end;
  return (.);
endsub;
```

```
/*中序遍历*/
function iterativeInOrder(name $, id, value[*], pos);
  outargs value;
  outargs pos;
  if exist(name) then do;
    array x[1] / nosymbols;
    rc=read array(name,x);
    length=dim(x);
    maxsize=x[1];

    leftChildId=BTreeGetChildId(name, id, 0);
    if leftChildID ^=. then rc=iterativeInOrder(name , leftChildId,
      value, pos);

    if 0< id & id<= length then do;
      value[pos]=x[id+1];
      pos=pos+1;
    end;

    rightChildId=BTreeGetChildId(name, id, 1);
    if rightChildID ^=. then rc=iterativeInOrder(name ,
      rightChildId, value, pos);
  end;
  return (.);
endsub;

/*后序遍历*/
function iterativePostOrder(name $, id, value[*], pos);
  outargs value;
  outargs pos;
  if exist(name) then do;
    array x[1] / nosymbols;
    rc=read_array(name,x);
    length=dim(x);
    maxsize=x[1];

    leftChildId=BTreeGetChildId(name, id, 0);
    if leftChildID ^=. then rc=iterativePostOrder(name ,
      leftChildId, value, pos);

    rightChildId=BTreeGetChildId(name, id, 1);
    if rightChildID ^=. then rc=iterativePostOrder(name ,
      rightChildId, value, pos);

    if 0< id & id<= length then do;
      value[pos]=x[id+1];
      pos=pos+1;
    end;
  end;
  return (.);
endsub;

/*广度优先遍历*/
function iterativeLayerOrder(name $, id, value[*], pos);
  outargs value; outargs pos;
  head 1; tail=1;

  array BTree[1] / nosymbols;
  call dynamic array(BTree, dim(value));

  if id . then return ( 1);
```

```
    BTree[head]=id;
    tail=tail+1;

    do while(head < tail);
      node=BTree[head];

      leftChildId=BTreeGetChildId(name, node, 0);
      if leftChildID ^=. then do;
        BTree[tail]=leftChildId;
        tail=tail+1;
      end;

      rightChildId=BTreeGetChildId(name, node, 1);
      if rightChildID ^=. then do;
        BTree[tail]=rightChildId;
        tail=tail+1;
      end;

      value[pos]= BTreeGetNodeValue(name, node);
      pos=pos+1;
      head=head+1;
    end;
    return (.);
  endsub;

  /*返回二叉树的大小*/
  function BTreeSize(name $);
    array y[1] / nosymbols;

    if exist(name) then do;
      array x[1] / nosymbols;
      rc=read_array(name,x);
      length=dim(x);
      rc=length-1;
    end;
    else do;
      rc=0;
    end;
    return (rc);
  endsub;
run;
quit;
```

程序 15-14 对 FCMP 版的 BTree 实现进行测试，测试结果与 PROTO 实现输出相同。

程序15-14 FCMP 版BTree 的简单测试程序

```
option cmplib=(work.funcs);
data _null_;
  rc=BTreeDefine('t', 10);

  array a[10] (1 2 3 4 5 6 7 8 9 10);
  rc=BTreeBuild('t',a );

  array b[10];
  rc=iterativePreOrder('t',1, b, 1 );
  put "NOTE: 前序遍历 " b[*]=;

  rc=iterativeInOrder('t',1, b, 1 );
  put "NOTE: 中序遍历 " b[*]=;
```

```
  rc iterativePostOrder('t',1, b, 1 );
  put "NOTE: 后序遍历 " b[*]=;

  rc=iterativeLayerOrder('t',1, b, 1 );
  put "NOTE: 广度优先 " b[*]=;
run;
```

系统输出如下：

```
WARNING:  Change BTree t (maxsize= 10 )
NOTE: Build 1 2 3 4 5 6 7 8 9 10 >> BTree t
NOTE: 前序遍历 b1=1 b2=2 b3=4 b4=8 b5=9 b6=5 b7=10 b8=3 b9=6 b10=7
NOTE: 中序遍历 b1=8 b2=4 b3=9 b4=2 b5=10 b6=5 b7=1 b8=6 b9=3 b10=7
NOTE: 后序遍历 b1=8 b2=9 b3=4 b4=10 b5=5 b6=2 b7=6 b8=7 b9=3 b10=1
NOTE: 广度优先 b1=1 b2=2 b3=3 b4=4 b5=5 b6=6 b7=7 b8=8 b9=9 b10=10
```

15.3　二叉树应用：算术表达式求值

二叉树在数据查找中具有广泛应用，它查找一个数据的时间复杂度为 $\log(n)$，而普通的数组或线性列表则为 n，因此二叉树在很多数据查找领域都有应用。二叉树在复杂问题分解方面非常有用，在计算机语言本身的编译处理中就会用到这一数据结构，如用于算术表达式求值中构建抽象语法树（AST）。

下面以设计一个简单的表达式求解器为例，看如何利用二叉树对一个给定算术表达式进行求值。为演示目的，我们假定该求解器只支持＋、－、×、÷ 和 ^ 乘方 5 种运算，数值也仅支持个位数 0~9，如表达式：1+2×3-8/4+5^2 。

要实现这样一个表达式求解器，首先要将复杂表达式分解到最细粒度。抽象看来，最简单的求值不外乎是操作数（通常是两个）和操作符（运算）的组合，即结果值 = *操作数 1* 操作符 *操作数 2*。比如，上面的表达式中，其正确的计算序列应该为 2×3 = 6, 8/4=2, 5 ^ 2 = 25； 最终表达式为 1+6-2+25=30。

同时，我们知道运算符本身还具有优先级的问题：先乘除，后加减，而乘方运算则更加优先。如果是同样的优先级，则可以自左向右方向进行结合求值。因此，表达式求值就是按照优先级从高往低的方向，对二元表达式进行求值；然后将计算结果代入次一级的二元运算直到整个表达式被计算完毕。则上面的表达式 1+2×3-8/4+5^2 可以分解为

- 5 ^ 2 =25
- 2 × 3 =6　8 / 4 =2
- 1 + 6 = 7　7 – 2 =5　5 + 25 =30

如果我们能够用构造如图 15-2 所示的二叉树表示，则表达式的求值可以采用由底至顶不断求解，直到最终得到根节点的值为止。

```
      if (root->leftChild != 0)
        v1=evaluatePostOrder( root->leftChild);
      if (root->rightChild!=0)
        v2=evaluatePostOrder( root->rightChild);

      if (c==43 ) return v1+v2; /*加法运算*/
      if (c==45 ) return v1-v2; /*减法运算*/
      if (c==42 ) return v1*v2; /*乘法运算*/
      if (c==47 ) return v1/v2; /*除法运算*/
      if (c==94 ) return pow((double)v1, (double)v2); /*乘方运算*/
      return(c-48); /*数字 0-9，十六进制 48-57*/
    }
  externcend;
run;
```

基于上面的 PROTO 二叉树结构和后序遍历求值函数，就可用 FCMP 函数来实现表达式求解函数 evaluate，该函数唯一参数为字符串表达式。主要算法就是根据基本准则动态构建语法树，然后使用后序遍历顺序进行求值；其中还用到一个对运算符的优先级进行判断的辅助函数 getlevel。实际上，由于 FCMP 中不支持将结构体 *struct* 作为函数参数，因此后序遍历求值必须实现在 PROTO 过程步中，而非 FCMP 过程步中。完整实现代码如程序 15-16 所示。

程序15-16　FCMP 实现表达式求解函数evaluate() 和优先级辅助函数 getlevel

```
proc fcmp library=work.funcs outlib=work.funcs.exp;
  /* 返回运算符 c 的优先级:1-3，越大表示优先*/
  function getlevel (c $);
    if c in ("+" "-") then do;
      return (1);
    end;
    else if c in ("/" "*") then do;
      return (2);
    end;
    else if c in ("^") then do;
      return (3);
      end;
    return(0);
  endsub;

  /* 表达式求解入口：比如rc=evaluate("1+2*3-8/4+5^2");*/
  function evaluate(exp $);
    struct BTree root;
    struct BTree lastnode;

    bstart=0;
    pos=1;
    do while (pos<= length(exp));
      c=substr(exp, pos,1);
      if c ^=" " then do;
        struct BTree node;
        node=create(rank(c));

        if bstart=0 then do;
          root=node;
          bstart=1;
        end;
        else do;
          if left(trim(c)) in ("+" "-" "*" "/" "^") then do;
```

```
              if  trim(left( byte( root.value)))
                in ("+" " " "*" "/" "^") then do;
                /*构造二叉树准则*/
                if getlevel( c ) > getlevel( byte(root.value) ) then do;
                  struct BTree orignode;
                  orignode=root.rightChild;
                  root.rightChild=node;
                  node.leftChild=orignode;
                end;
                else do;
                  node.leftChild=root;
                  root=node;
                end;
              end;
              else do;
                node.leftChild=lastnode;
                root=node;
              end;
            end;
            else do;
              lastnode.rightChild=node;
            end;
          end;
          lastnode=  node ;
        end;
        pos=pos+1;
      end;
      /*后序遍历顺序求值*/
      result=evaluatePostOrder(root);
      return(result);
    endsub;
run;
quit;
```

至此，我们就可以在 FCMP 或 DATA 步中对函数 evaluate 进行调用。用户只需要将需要计算的算术表达式传递给它即可。由于本例子只是为了说明二叉树的应用，仅定义了有限的数据和操作符。在 PROC FCMP 中调用示例见程序 15-17，而在 DATA 步中调用示例如程序 15-18 所示。

程序15-17 FCMP 中使用表达式解析器

```
proc fcmp library=work.funcs  ;
  exp="1+2*3-8/4+5^2";
  result=evaluate(exp);
  put "表达式：" exp "=" result;
run;
quit;
```

程序15-18 DATA 步中使用表达式解析器

```
options cmplib=work.funcs;
data _null_;
  exp="1+2*3-8/4+5^2";
  result=evaluate(exp);
  put "表达式：" exp "=" result;
run;
```

这两个程序运算后都能输出正确的答案，感兴趣的读者可以在此基础上构建更加强大的表达式求解器。

第16章

数据结构——矩阵运算

矩阵是一个由若干行列元素组成的矩形阵列，当行数与列数相等时称为方块矩阵，简称方阵。只有一行或一列的矩阵称为行矩阵或列矩阵，它等价于向量。矩阵作为线性代数和数值分析的基础数学工具，在统计分析、物理学计算、计算机图形学等领域具有非常广泛的应用，最常见的应用为线性 / 非线性方程组求解、几何空间的线性变换等。

概率论中也常用矩阵表示概率转移矩阵和发射矩阵，用于定义有限概率空间中的马尔可夫链；描述性统计中则常用协方差矩阵表示若干随机变量之间的协方差关系，而在线性回归分析的最小二乘法中用于表述样本之间的线性关系。矩阵运算可以说是数据分析的基础，理解并掌握它是迈向数据分析核心的重要一步。矩阵的基本运算主要包括如下内容。

（1）加减运算：相同大小的两个矩阵相应位置的元素进行加减，称为矩阵的和或差，记为 $\boldsymbol{A}\pm\boldsymbol{B}$，它满足交换律 $(\boldsymbol{A}+\boldsymbol{B})-\boldsymbol{C}=\boldsymbol{A}+(\boldsymbol{B}-\boldsymbol{C})$。

（2）数乘运算：一个标量 k 乘以矩阵 $\boldsymbol{A}$ 每个元素，形成一个新的矩阵，称为矩阵的积，记为 $k\boldsymbol{A}$，它满足交换律和分配律。

（3）矩阵乘法：当第 1 个矩阵的列数跟第 2 个矩阵的行数相等时，矩阵之间可以相乘。比如，m 行 n 列矩阵 $\boldsymbol{A}$ 乘以 n 行 p 列矩阵 $\boldsymbol{B}$，得到的乘积矩阵 $\boldsymbol{AB}$ 为 m 行 p 列的矩阵。其中矩阵 $\boldsymbol{AB}$ 的第 i 行 j 列元素等于矩阵 $\boldsymbol{A}$ 的第 i 行元素与矩阵 $\boldsymbol{B}$ 第 j 列对应元素的乘积之和：

$$[\boldsymbol{AB}]_{i,j}=\boldsymbol{A}_{i,1}\boldsymbol{B}_{1,j}+\boldsymbol{A}_{i,2}\boldsymbol{B}_{2,j}+\cdots+\boldsymbol{A}_{i,n}\boldsymbol{B}_{n,j}=\sum_{c=1}^{n}\boldsymbol{A}_{i,c}\boldsymbol{B}_{c,j}$$

矩阵乘法不满足交换律，即 $\boldsymbol{AB}\neq\boldsymbol{BA}$。矩阵乘法规则看起来比较奇怪，但它却是表述线性方程式变量之间关系的绝妙发明，在矩阵有关的运算中非常常见。

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}\times\begin{bmatrix}1 & 4\\2 & 5\\3 & 6\end{bmatrix}=\begin{bmatrix}14 & 32\\32 & 77\end{bmatrix}$$

（4）矩阵转置：将矩阵的行列进行互换，原来 i 行 j 列的元素变为 j 行 i 列元素，行列数相应变化，是初等变换中的位置变换。

（5）逆矩阵：对于 n 阶方阵 $\boldsymbol{A}$，如果存在 $\boldsymbol{AC}=\boldsymbol{CA}=\boldsymbol{E}$，其中 $\boldsymbol{E}$ 为 n 阶单位矩阵（仅对角线为 1，其余为 0 的矩阵），则称方阵 $\boldsymbol{C}$ 是方阵 $\boldsymbol{A}$ 的逆矩阵，记为 $\boldsymbol{C}=\boldsymbol{A}^{-1}$。可逆矩阵只有唯一的逆矩阵，可逆矩阵有满秩且满足行列式 $\det\boldsymbol{A}\neq 0$ 的性质。

（6）矩阵还有其他初等变换，如倍乘变换和倍加变换。比如，矩阵某一行（或列）

乘以非零 k 的倍乘变换，以及某一行（或列）乘以非零 k 并累加到另一行（或列）上的倍加变换。

计算机语言中通常用二维数组表示矩阵元素，也有的计算机语言会提供向量或者矩阵对象来封装矩阵运算。SAS 语言作为面向数据分析的语言，在不同层面上实现了对矩阵运算的支持，主要包括以下几种方法。

（1）DATA 步中可利用二维数组直接进行矩阵运算，所有矩阵运算规则需要用户自己实现，较麻烦。

（2）在 FCMP 过程步中利用 FCMP 系统例程进行。由于 DATA 步可以调用用户自定义 FCMP 函数，因此 DATA 步中可重用经过用户再封装的 FCMP 系统例程和函数。

（3）在 DS2 过程步中使用系统包 MATRIX 进行矩阵运算。

（4）购买 SAS/IML 专业矩阵运算包的软件许可，IML 全称是 Interactive Matrix Language，它包含上百个专门分析和算法实现，涵盖矩阵、时间序列、数值分析、优化算法、数值模拟和数据可视化等各领域。

16.1 FCMP 矩阵运算

PROC FCMP 过程步提供 FCMP 子系统的数组跟 SAS 数据集之间转换函数 READ_ARRAY 和 WRITE_ARRAY，它们使 FMCP 数组数据能够方便地输出到 SAS 数据集，供其他 SAS 程序、宏或过程步作进一步处理，而它们处理的结果数据集也可以反过来构造 FCMP 数组在 FCMP 函数内作进一步处理。值得一提的是 FCMP 子系统还支持用 RUN_MACRO 函数来执行一个预定义宏以及用 RUN_SASFILE 函数来执行一个外部 SAS 文件。

由于 FCMP 内置系统函数能将一维数组 / 二维数组当作矩阵进行常见的矩阵运算，而 DATA 步中又可以很方便地重用这些函数，因此 FCMP 矩阵运算为日常数据分析中的矩阵运算提供了简便方法。但 FCMP 矩阵运算中除 ZeroMatrix、FillMatrix 和 Identity 例程外通常不支持缺失值处理，功能相对有限。FCMP 矩阵运算的主要功能为：

1. 矩阵操作——赋值

（1）将矩阵所有元素清零。

```
call zeromatrix(X);
```

参数：

- X 输入/输出参数，为任意m行n列的矩阵。

（2）将矩阵所有元素设置为特定值。

```
call fillmatrix(X, s);
```

参数：

- X 输入/输出参数，为任意m行n列的矩阵。

- s 输入参数，给定标量。

2. 矩阵操作——加减法、元素乘法、矩阵乘法

（1）矩阵加法：如果矩阵大小相等，对应元素相加；如果 X 或 Y 为标量，则矩阵中的每个元素都加上标量。

```
call addmatrix(X, Y, Z);
```

参数：

- X 或Y 输入参数，为m 行 n 列的矩阵或者一个标量s。
- Z 输出参数，为 m行n列的矩阵。Z=X+Y，Z[m, n]=X[m, n]+Y[m, n]。

（2）矩阵减法：如果矩阵大小相等，对应元素相减；如果 X 或 Y 为标量，则矩阵中的每个元素都减去标量。

```
call subtractmatrix(X, Y, Z);
```

参数：

- X 输入参数，m行n列的矩阵或标量，X[m,n]。
- Y 输入参数，m行n列的矩阵或标量，Y[m,n]。
- Z 输出参数，m行n列的矩阵，Z[m,m]，Z=X－Y。

（3）元素乘法——如果矩阵大小相等，对应元素相乘。如果 X 或 Y 其中一个为标量，则矩阵中的每个元素乘以标量，返回一个矩阵。如果 X 和 Y 都是标量，则返回一个标量。但需要注意 FCMP 矩阵元素乘法不支持行矩阵、列矩阵相乘，而在 DS2 中则可以。

```
call elemmult(X, Y, Z);
```

参数：

- X 输入参数，m行n列的矩阵。
- Y 输入参数，m行n列的矩阵。
- Z 输出参数，m行n列的矩阵。

（4）矩阵乘法：计算两个输入矩阵的乘积，它要求第一个矩阵的列数 n 等于第二个矩阵的行数 n，输出矩阵的行数为第 1 个矩阵的行数 m，列数为第 2 个矩阵的列数 p。

```
call mult(X, Y, Z);
```

参数：

- X 输入参数，m 行 n 列的矩阵，X[m,n]。
- Y 输入参数，n 行 p 列的矩阵，Y[n, p]。
- Z 输入参数，m 行 p 列的矩阵，Z[m, p]，其中Z[m,p]=X[m,n] × Y[n,p]。

3. 矩阵操作——转置

矩阵转置：将矩阵对应的行变成对应的列。

```
call transpose(X, Y);
```

参数：

- X 输入参数，m行n列的矩阵，X[m,n]。
- Y 输出参数，n行m列的矩阵，Y[m,n]，即 Y = X' 。

4. 方阵操作——指数矩阵和乘幂

（1）将给定方块矩阵 X 乘以标量 t 并变换为以 e 为底的指数矩阵 e^{tX}。SAS 采用 Golub 和 van Loan（1989）中提到的 Padé 近似算法，该算法不会对每个元素取幂。

```
call expmatrix(X, t, Y);
```

参数：

- X 输入参数，m行m列的方块矩阵。
- t 输入参数，浮点标量。
- Y 输出矩阵，m行m列的方块矩阵，其中元素关系为 $Y = e^{tX}$。

（2）将方块矩阵变换到指定整数幂。

```
call power(X, a, Y);
```

参数：

- X 输入参数，m行m列的方块矩阵，X[m,m] 。
- a 输入参数，整数标量，即整数幂 如果输入参数a小于0，则取0；如果不是整数，SAS默认自动取整。
- Y 输出参数，m行m列的方块矩阵，Y[m,m] ，$Y= X^a$。

5. 方阵操作——求行列式、逆矩阵、单位矩阵、楚列斯基分解

（1）求方阵行列式。如果对应的行列式 $\det A=0$ 说明该矩阵是奇异矩阵、不可逆；否则为非奇异矩阵、可逆且满秩。

```
call det(X, a);
```

参数：

- X 输入参数，m行m列的方块矩阵。
- a 输出参数，返回矩阵的 a=|X|。

需要特别注意：

①特征值的乘积是一个标量，如果矩阵行列式为 0，则该矩阵是奇异矩阵，没有逆矩阵。该方法执行 LU 分解并收集对角线的乘积。

②当方块矩阵每个特征值都大于等于 0，则该矩阵为半正定；当所有特征值都大于 0 时，则该矩阵为正定。

③当且仅当其行列式不为零，则方块矩阵是非奇异的。方块矩阵奇异当且仅当其代表的线性变换是自同构的。

④如果 A 是奇异矩阵，则 A 必不可逆，此时 $AX=0$ 有无穷解，$AX=b$ 有无穷解或者无解。如果 A 是非奇异矩阵，则 A 必可逆，此时 $AX=0$ 有且只有唯一零解，$AX=b$ 有唯一解。

（2）方块矩阵的逆矩阵，输入矩阵必须是非奇异矩阵。

```
call inv(X, Y);
```

参数：

- X 输入参数，m 行 m 列的方块矩阵。
- Y 输出参数，m行m列的方块矩阵，其中Y[m,m]=X' [m, m]，其中满足XY=YX=I，其中I为单位矩阵。

（3）将指定方阵变换为对应的单位矩阵。即主对角线上值为 1，其余值为 0 的 m 阶单位矩阵。单位矩阵的特征值为 1，其主对角线上的特征值之积（即行列式）等于 1，其主对角线上的特征值之和（即迹）等于 m。

```
call identity(X);
```

参数：

- X 输入输出参数，m行m列的方块矩阵。

（4）计算方块矩阵的楚列斯基（Cholesky）分解。要求方阵是为对称且正定，其中对称是指上下三角对称。如果 X 不是对称正定，则结果矩阵 Y 会用缺失值填充。

```
call chol(X, Y, v);
```

参数：

- X输入参数，要求是对称、正定的方块矩阵 X [m, m]。
- Y输出参数，指定m行m列的方块矩阵，用于输出Cholesky 分解项 Y[m, m]为具有严格正对角条目的下三角矩阵，Y *表示方阵Y的共轭转置。
- v 输入参数，可选。0表示检查矩阵是否对称（默认）；1则假定矩阵对称，运算更快。

程序 16-1 演示了以上 FCMP 矩阵运算的完整功能，读者可检查结果来了解各方法功能。

程序16-1　FCMP矩阵处理功能示例

```
proc fcmp ;
  array ma_2_3[2,3] (
    1, 2, 3,
    4, 5, 6);
  array ma_3_2[3,2] (
    1, 2,
    3, 4,
    5, 6);
  array ma_3_3[3,3] ( 0.30, -0.78, -0.82,
                      0.54,  1.74,  1.2,
                      1.30,  0.25,  1.49);
  array mb_2_3[2,3] (
    7,   8,   9,
    10, 11, 12);

  array mc_3_3[3,3] (
    2, 2, 3
    2, 4, 2
    3, 2, 6);
```

```
array mo_2_3[2,3];
array mo_2_2[2,2];
array mo_3_3[3,3];

put "矩阵清零";
call zeromatrix(ma_3_2);
do i=1 to dim(ma_3_2);
  put ma_3_2[i, 1] best.2 ma_3_2[i, 2] best.2;
end;

put "矩阵赋值";
call fillmatrix(ma_3_2, 99);
do i=1 to dim(ma_2_3);
  put ma_3_2[i, 1] best.2 ma_3_2[i, 2] best.2;
end;

put "矩阵A";
do i=1 to dim(ma_2_3);
  put ma_2_3[i, 1] best.2 ma_2_3[i, 2] best.2  ma_2_3[i, 3] best.2;
end;
put "矩阵B";
do i=1 to dim(mb_2_3);
  put mb_2_3[i, 1] best.2 mb_2_3[i, 2] best.2  mb_2_3[i, 3] best.2;
end;

put "矩阵加法：矩阵+矩阵（等大小）";
call addmatrix(ma_2_3, mb_2_3, mo_2_3);
do i=1 to dim(mo_2_3);
  put mo_2_3[i, 1] best.2 mo_2_3[i, 2] best.2  mo_2_3[i, 3] best.2;
end;
put "矩阵加法：矩阵+标量";
call addmatrix(ma_2_3, 10, mo_2_3);
do i=1 to dim(mo_2_3);
  put mo_2_3[i, 1] best.2 mo_2_3[i, 2] best.2  mo_2_3[i, 3] best.2;
end;

put "矩阵减法：矩阵-矩阵（等大小）";
call subtractmatrix(ma_2_3, mb_2_3, mo_2_3);
do i=1 to dim(mo_2_3);
  put mo_2_3[i, 1] best.2 mo_2_3[i, 2] best.2  mo_2_3[i, 3] best.2;
end;
put "矩阵减法：矩阵-标量";
call subtractmatrix(ma_2_3, 10, mo_2_3);
do i=1 to dim(mo_2_3);
  put mo_2_3[i, 1] best.2 mo_2_3[i, 2] best.2  mo_2_3[i, 3] best.2;
end;

put "矩阵元素乘法：矩阵×矩阵（等大小）";
call elemmult(ma_2_3, mb_2_3, mo_2_3);
do i=1 to dim(mo_2_3);
  put mo_2_3[i, 1] best.2 mo_2_3[i, 2] best.2  mo_2_3[i, 3] best.2;
end;
put "矩阵元素乘法：矩阵×标量";
call elemmult(ma_2_3, 10, mo_2_3);
do i=1 to dim(mo_2_3);
  put mo_2_3[i, 1] best.2 mo_2_3[i, 2] best.2  mo_2_3[i, 3] best.2;
end;

put "矩阵乘法：";
call mult(ma_2_3, ma_3_2, mo_2_2);
do i=1 to dim(mo_2_2);
```

```
    put mo_2_2[i, 1] best.2 mo_2_2[i, 2] best.2;
  end;

  put "矩阵转置";
  array mo_3_2 [3,2];
  call transpose(ma_2_3, mo_3_2);
  do i=1 to dim(mo_3_2);
    put mo_3_2[i, 1] best.2 mo_3_2[i, 2] best.2;
  end;

  put "方形矩阵 变换 Y=e^tX";
  call expmatrix(ma_3_3, 3, mo_3_3);
  do i=1 to dim(mo_3_3);
    put mo_3_3[i, 1] best.2 mo_3_3[i, 2] best.2  mo_3_3[i, 3] best.2;
  end;

  put "方形矩阵 变换 Y=X^a";
  call power(ma_3_3, 3, mo_3_3);
  do i=1 to dim(mo_3_3);
    put mo_3_3[i, 1] best.2 mo_3_3[i, 2] best.2 mo_3_3[i, 3] best.2;
  end;

  put "方形矩阵求行列式 DET";
  call det(ma_3_3, ret);
  put ret=;

  put "方形矩阵变换逆矩阵";
  call inv(ma_3_3, mo_3_3);
  do i=1 to dim(mo_3_3);
    put mo_3_3[i, 1] best.2 mo_3_3[i, 2] best.2  mo_3_3[i, 3] best.2;
  end;
  call inv(mo_3_3, mo_3_3); /*二次逆即得原矩阵*/

  put "方形矩阵单位矩阵";
  call identity(mo_3_3);
  do i=1 to dim(mo_3_3);
    put mo_3_3[i, 1] best.2 mo_3_3[i, 2] best.2  mo_3_3[i, 3] best.2;
  end;

  put "方形矩阵 Cholesky（要求矩阵是对称，正定）";
  call chol(mc_3_3, mo_3_3, 0);
  do i=1 to dim(mo_3_3);
  put mo_3_3[i, 1] best.2 mo_3_3[i, 2] best.2  mo_3_3[i, 3] best.2;
  end;
run;
quit;
```

系统输出如图 16-1 所示。

既然 FCMP 子系统中已经有这些矩阵函数，我们是否在 DATA 步中可以直接调用呢？很遗憾，如果直接调用 SAS 系统会报告如下错误：

```
ERROR 251-185: 子程序 MULT 未知，或无法访问。请查看您的拼写。它在可执行图像的路径中找不到，或存在不正确或缺失的子程序描述符信息。
```



图 16-1　基于 FCMP 函数的矩阵运算

为了在 DATA 步中重用这些矩阵函数，而不是自己重新发明轮子，我们需要利用自定义 FCMP 函数来桥接 DATA 步和 FCMP 系统例程的程序空间。比如矩阵乘法可封装为 FCMP 函数 m_mult（见程序 16-2）。

程序16-2　使用FCMP 自定义函数将矩阵处理封装给 DATA步使用

```
proc fcmp outlib=work.funcs.math;
  subroutine m_mult(a[*,*], b[*,*], c[*,*]);
    outargs c;
    call mult(a, b, c);
  endsub;
run;
quit;
```

这样就可在 DATA 步中重用该矩阵函数了（见程序 16-3），可以根据需要在 FCMP 中扩展任意矩阵运算操作供 DATA 步使用。

程序16-3　如何在DATA步中使用FCMP 矩阵处理功能

```
options cmplib=work.funcs;
data _null_;
  array ma_2_3[2,3] (1, 2, 3, 4, 5, 6);
  array ma_3_2[3,2] (1, 2, 3, 4, 5, 6);
  array mo_2_2[2,2];
```

```
  /*call mult(ma_2_3, ma_3_2, mo_2_2); 直接调用会失败*/
  call m_mult(ma_2_3, ma_3_2, mo_2_2);

  do i=1 to 2;
    put "(" mo_2_2[i,1] 3."," mo_2_2[i, 2] 3. ")";
  end;
run;
quit;
```

系统输出：

```
(22, 28)
(49, 64)
```

16.2　DS2 矩阵运算

作为 DATA 步的第二代语言，DS2 对矩阵的支持更加丰富；DS2 采用包程序 MATRIX 将矩阵运算进行封装，采用对象引用的方式进行调用。比如：

```
dcl package matrix m1(2,3);
dcl package matrix m2(3,2);
dcl package matrix m3;
m3=m1.mult(m2);
```

DS2 的包程序 MATRIX 对象封装了如下方法，我们很容易进行高级矩阵运算，表 16-1 列出了 DS2 MATRIX 对象支持的所有成员方法。

表 16-1　DS2 Matrix 对象的矩阵运算函数

运算类别	方法 / 函数	说　明
一元运算	 INVERSE DET TRANS EXP LOG SQRT ABS FLOOR COPY	一元运算作用于每个元素，执行在单个矩阵上；矩阵必须是方块矩阵或奇异矩阵 • INVERSE 矩阵求逆，要求矩阵为方块矩阵 • DET 矩阵求行列式，要求矩阵为方块矩阵 • TRANS 矩阵转置 • EXP 指数值 • LOG 取自然对数 • SQRT 为开方运算 • ABS 取绝对值 • FLOOR 向下取整 • COPY 复制整个矩阵
二元算术	ADD SUB MULT	• 加法 / 减法：两个矩阵的行列必须相同，或者第二个矩阵为 1x1 矩阵，即按照标量处理；遇到缺失值不会显式报告错误 • 矩阵乘法：第一个矩阵的列和第二矩阵的行必须相等，遇到缺失值会报运行时错误

（续表）

运算类别	方法 / 函数	说　明
二元关系	AND OR	• 矩阵元素对应位置进行二元关系比较，结果为由 0 或 1 组成的矩阵
以下运算返回单个标量，而非矩阵		
ANY 关系	ANY_EQ ANY_GE ANY_GT ANY_LE ANY_LT ANY_NE	• 矩阵大小必须相等，对应位置进行比较，结果为标量 0 或 1 组成的矩阵。分别表示等于、大于等于、大于、小于等于、小于、不等于关系
ANY 逻辑	ANY_AND ANY_OR	• 任何一个元素逻辑比较为真，则返回 1，否则为 0
ALL 逻辑	ALL_AND ALL_OR	• 矩阵所有对应元素的比较都是 1，则返回 1，否则为 0
元素操作	EMULT EDIV EPOW EMOD EMIN EMAX	• 适用于一般矩阵，第二个矩阵可以是相同大小的矩阵，也可以是行数（或列数）相等的行矩阵（或列矩阵），或者是 1x1 矩阵（标量） • 对每个对应元素逐一进行运算，包括乘、除、指数、除余、最小值、最大值等。
生命周期	创建 / 初始化 DELTET	dcl package matrix m([rows, cols]); 或者 dcl package matrix m; m=_new_ matrix(2, 2); • 删除矩阵对象实例
	ROWS COLS	• 返回矩阵行数 • 返回矩阵列数
数组交互	IN OUT	• 加载数组到矩阵中，支持标准数组和变量数组（VARARRAY） • 输出矩阵到数组中，也可用第 2 个参数指定矩阵的目标行或来源行，DS2 数组可进一步用 OUTPUT 语句输出到 SAS 数据集
	TOARRAY TOVARARRAY	• 将整个矩阵数据输出到标准数组 • 将整个矩阵数据输出到变量数组（VARARRAY）

DS2 矩阵使用构造器创建时会自动填充 0 值而非缺失值。对于空值或缺失值，矩阵加减法并不报错，而矩阵乘法则会报运行时错误；当出现除零错误时，SAS 日志一般不报告错误而是直接返回缺失值。对于 DS2 线程程序中定义的矩阵对象，每个线程有自己的矩阵实例，MATRIX 对象不支持在节点或线程上进行数据分区来实现并行矩阵运算，每个线程只在自己的 MATRIX 对象实例上进行工作。

下面举例说明 DS2 中矩阵操作，如矩阵数据的读写。通常需要使用 IN/OUT 方法从数据集中读取数据，运算后写回。大多数矩阵运算都需要使用类似的逻辑，并保存结果到 SAS 数据集中。程序 16-4 展示了 DS2 矩阵操作的基本用法。

程序16-4　DS2 矩阵操作示例,输出逆矩阵和单位矩阵

```
data A; /*生成 4x4 的数据集，变量为 x1 x4*/
  array x[4];
  infile datalines;
  input x1-x4;
  datalines;
 1 5 2 3
 3 3 1 7
 2 3 8 9
 3 6 7 4
run;
proc print data=A;
  title "原始矩阵";
run;

proc ds2;
  data B(keep=(y1 y2 y3 y4)) E(keep=(z1 z2 z3 z4))/overwrite=yes;
    vararray double x[4];
    dcl package matrix m;
    dcl package matrix m_i;
    dcl package matrix m_e;
    dcl double a[4, 4];
    dcl int r c;
    vararray double y[4];
    vararray double z[4];

    method init();
      m=_new_ [this] matrix(4, 4);
      r=1;
    end;

    method run();
      set A;        /*指定输入数据集*/
      m.in(x, r); /*从输入数据集中读取一行，放到矩阵 r 行中*/
      r + 1;
    end;

    method term();
      m_i=m.inverse();  /* 矩阵数据已经准备好，求逆矩阵*/
      do r=1 to 4;      /* 借助全局变量数组 y 输出逆矩阵 B*/
        m_i.out(y, r);
        output B;
    end;

    m_e=m.mult(m_i); /* 生成单位矩阵 E*/
    do r=1 to 4;     /* 借助全局变量数组 z 输出4阶单位矩阵 E*/
      m_e.out(z, r);
      output E;
    end;

    m_i.toarray(a);       /* 输出矩阵 m_i 到数组 a 并打印到日志 */
    do r=1 to 4;
      put a[r, 1]  7.2 a[r, 2]  7.2 a[r, 3]  7.2 a[r, 4]  7.2;
    end;
  end;
enddata;
run;
quit;

/*打印结果矩阵*/
proc print data B;
```

```
  title "逆矩阵";
run;

proc print data=E;
  title "单位矩阵";
run;
```

如果不使用自带隐性循环的系统函数 RUN，则可以在 INIT 函数中自己构造循环来完成矩阵数据的加载（见程序 16-5）。

程序16-5　在 INIT()系统方法中初始化矩阵数据

```
method init();
  m=_new_ [this] matrix(4, 4);
  do r=1 to 4;
    set A;
    m.in(x,r);
  end;
end;
```

16.3　矩阵应用：线性方程组求解

中国古代算书《孙子算经》中提出著名的鸡兔同笼问题，其原文为：今有雉兔同笼，上有三十五头，下有九十四足；问雉兔各几何。现在我们知道该问题可以归结为二元一次线性方程进行求解，即假定 x 为鸡，y 为兔，则 $x+y=35$，$2x+4y=94$，求 $x=$？ $y=$？的问题。

现在要用 SAS 编程来求解给定的线性方程组。我们知道一元一次方程 $ax=b$ 的解是利用 a 的逆运算 a^{-1} 去乘两边，得到 $ax/a=b/a$，即 $x=b/a$。同理，多元线性方程组 $\boldsymbol{AX}=\boldsymbol{B}$ 也可采用相同的思路求解，其中 $\boldsymbol{AX}=\boldsymbol{B}$ 为方程组的向量表达，$\boldsymbol{A}$ 为 n 阶矩阵，$\boldsymbol{X}$、$\boldsymbol{B}$ 为列向量。只不过矩阵 $\boldsymbol{A}$ 采用的逆运算为矩阵求逆 $\boldsymbol{A}^{-1}$ 而不是标量的倒数（**注**：矩阵乘法不满足交换律不能写成 $1/\boldsymbol{A}$），然后 $\boldsymbol{A}^{-1}$ 左乘 $\boldsymbol{B}$ 即可得到列向量 $\boldsymbol{X}$ 的值。

比如，上面鸡兔同笼问题的方程 $x+y=35$，$2x+4y=94$ 写成矩阵形式为

$$\begin{vmatrix}1 & 1\\2 & 4\end{vmatrix}\begin{vmatrix}x\\y\end{vmatrix}=\begin{vmatrix}35\\94\end{vmatrix}$$

要求解向量$\boldsymbol{X}$ $\begin{vmatrix}x\\y\end{vmatrix}$，只需将左边的系数矩阵$\begin{vmatrix}1 & 1\\2 & 4\end{vmatrix}$化掉即可，而化掉该矩阵只需等式两边乘以其逆矩阵$\begin{vmatrix}2 & -0.5\\-1 & 0.5\end{vmatrix}$即可。

$$\begin{vmatrix}2 & -0.5\\-1 & 0.5\end{vmatrix}\begin{vmatrix}1 & 1\\2 & 4\end{vmatrix}\begin{vmatrix}x\\y\end{vmatrix}=\begin{vmatrix}2 & -0.5\\-1 & 0.5\end{vmatrix}\begin{vmatrix}35\\94\end{vmatrix}$$

由上面的方程进一步化简可得：

$\begin{vmatrix}1 & 0\\0 & 1\end{vmatrix}\begin{vmatrix}x\\y\end{vmatrix}=\begin{vmatrix}2 & 0.5\\1 & 0.5\end{vmatrix}\begin{vmatrix}35\\94\end{vmatrix}$，即 $\begin{vmatrix}x\\y\end{vmatrix}=\begin{vmatrix}23\\12\end{vmatrix}$，故 $x=23$，$y=12$，即 23 只鸡和 12 只兔。

根据上面的分析写成 DS2 矩阵运算程序（见程序 16-6），其中矩阵求逆是通过矩阵的 inverse 方法求得。读者可修改参数来求解任意线性方程组。

程序16-6　求解线性方程组：鸡兔同笼问题

```
proc ds2  ;
  data dsI(keep=(y1 y2)) dsZ(keep=(z)) / overwrite yes;
    vararray double y[2];/*输出逆矩阵数据集 dsI 用*/
    dcl double z ;/*输出结果向量数据集 dsZ 用*/

    method init();
      dcl package matrix m_A;
      dcl double a[2,2];

      dcl package matrix m_B;
      dcl double b[1,2];

      dcl package matrix m_i;
      dcl package matrix m_X;
      dcl double x[ 2];
      dcl int i;

      a:=(1, 1, 2, 4);
      m_A=_new_ matrix(2,2);
      m_A.load(a);

      b:=(35, 94);
      m_B=_new_ matrix(2,1);
      m_B.load(b);

      m_i=m_A.inverse();/*矩阵求逆*/
      do i=1 to 2;
        m_i.out(y, i);output dsI;
      end;

      m_X=m_i.mult( m_B );
      m_X.toarray(x);
      do i=1 to 2;
        z=x[i];output dsZ;
      end;
    end;
  enddata;
run;
quit;
/*打印逆矩阵和结果向量*/
proc print data=dsi;run;
proc print data=dsz;run;
```

系统输出如图 16-2 所示，左右两个数据集分别为逆矩阵和结果向量。

Obs	y1	y2
1	2	-0.5
2	-1	0.5

Obs	z
1	23
2	12

图 16-2　矩阵的逆与结果向量

16.4 矩阵应用：非线性方程组求解

当某个方程中因变量与自变量的关系不是一次幂关系，而是平方关系、对数关系、指数关系或三角函数关系时，则该方程为非线性方程。只要方程组中有任何一个方程为非线性方程时，它们构成的方程组就是非线性方程组。数学上已知，非线性方程一般不能求得精确解，只能利用迭代法求近似解，其数学基础依然是矩阵运算。

现在以下面的非线性方程组为例，探讨如何用 SAS 程序求解非线性方程组的根 x_1 和 x_2。

$$x_1^2-10x_1+x_2^2+8=0$$
$$x_1x_2^2+x_1-10x_2+8=0$$

对于上面的非线性方程组，我们可将它记为 $f(x^k)=\begin{vmatrix} f_1(x_1,x_2,\cdots,x_n) \\ f_2(x_1,x_2,\cdots,x_n) \\ f_k(x_1,x_2,\cdots,x_n) \end{vmatrix}=0$ ，写成向量形式即为 $F(x)=0$。

假设 $x^k=(x_1^k, x_2^k, \cdots, x_n^k)^T$ 是 $F(x)=0$ 的第 k 次近似根，将函数 $F(x)$ 的每个分量 $f_1(x_1,x_2,\cdots,x_n)$，$f_2(x_1,x_2,\cdots,x_n)$，$f_n(x_1,x_2,\cdots,x_n)$ 在 $x^k=(x_1^k,x_2^k,\cdots,x_n^k)^T$ 处进行多元泰勒展开并取到一次项为止，则有

$$f(x)=f_i(x_1,x_2,\cdots,x_n)=f_i(x_i^k)+\frac{\partial f_i(x^k)}{\partial x_1^k}(x_1-x_1^k)+\cdots+\frac{\partial f_i(x^k)}{\partial x_n^k}(x_n-x_n^k)=0$$

写成向量形式为

$$F(x)\approx F(x^k)+F'(x^k)(x-x^k)=0$$

则根据牛顿法定义，若 $F'(x^k)$ 非奇，第 k 次近似根可根据如下公式得到，其中初始向量 x^0 为假设值。

$$x^{k+1}=x^k-F'(x^k)^{-1}F(x^k)\text{，其中 } k=0.1,\cdots$$

公式中 $F'(x^k)$ 为 x^k 处的雅可比矩阵，它是由方程的一阶偏导数按一定方式排列构成的矩阵，利用它可以对一个可微的方程和目标点进行最优线性逼近。

$$F'\left(x^k\right)\begin{bmatrix} \frac{\partial f_1\left(x^k\right)}{\partial x_1} & \frac{\partial f_1\left(x^k\right)}{\partial x_2} & \frac{\partial f_1\left(x^k\right)}{\partial x_n} \\ \frac{\partial f_2\left(x^k\right)}{\partial x_1} & \frac{\partial f_2\left(x^k\right)}{\partial x_2} & \frac{\partial f_2\left(x^k\right)}{\partial x_n} \\ \frac{\partial f_3\left(x^k\right)}{\partial x_1} & \frac{\partial f_3\left(x^k\right)}{\partial x_2} & \frac{\partial f_3\left(x^k\right)}{\partial x_n} \end{bmatrix}$$

计算时先人为假设初始近似解向量 x^0，算出 $F(x^k)$ 和 $F'(x^k)$，然后解牛顿线性方程组 $F'(x^k)\Delta x^k=-F(x^k)$ 得到 Δx^k，则新解为 $x^{k+1}=x^k+\Delta x^k$。然后用新解 x^{k+1} 逐次迭代计算出 x^1，x^2，$\cdots$，x^k（迭代 k 次）。数学上已证明若 $F'(x)$ 在解 x^* 附近连续，非奇异且 $F'(x)$ 在 x^* 邻域满足条件 $\left\|F'(x)-F'\left(x^*\right)\right\|\leq r\left\|x-x^*\right\|$，则我们就可不断迭代求得其近似解。

具体到上面的非线性方程组，假定自变量为 x_1、x_2，则定义两个函数：

$$f_1(x_1, x_2)=x_1^2-10x_1+x_2^2+8=0$$
$$f_2(x_1, x_2)=x_1x_2^2+x_1-10x_2+8=0$$

即

$$f\left(x^k\right)=\begin{vmatrix}x_1^2-10x_1+x_2^2+8\\ x_1x_2^2+x_1-10x_2+8\end{vmatrix}$$，其中 $k=1,2$，记为 $F(x)$。

对 $F(x)$ 的每个分量 $f_1(x_1, x_2)$、$f_2(x_1, x_2)$ 在 x^k 处分别对 x_1、x_2 求偏导数可得雅可比矩阵 $J(x^k)$：

$$F'\left(x^k\right)=\begin{bmatrix}\dfrac{\partial f_1\left(x^k\right)}{\partial x_1} & \dfrac{\partial f_1\left(x^k\right)}{\partial x_2}\\ \dfrac{\partial f_2\left(x^k\right)}{\partial x_1} & \dfrac{\partial f_2\left(x^k\right)}{\partial x_2}\end{bmatrix}=\begin{bmatrix}2x_1-10 & 2x_2\\ x_2^2+1 & x_1\times 2x_2-10\end{bmatrix}$$

即

$$J(x^k)=\begin{bmatrix}2x_1-10 & 2x_2\\ x_2^2+1 & x_1\times 2x_2-10\end{bmatrix}$$

由于 $J\left(x^k\right)=\begin{bmatrix}\Delta x_1\\ \Delta x_2\end{bmatrix}=f\left(x^k\right)$，其中左侧 $J(x^k)$ 和右侧 $f(x^k)$ 都可以从初始 ($k=0$) 人为假定的解 $\begin{bmatrix}x_1^0\\ x_2^0\end{bmatrix}$ 计算出，则我们就可以用反复迭代来逼近最终的解 $\begin{bmatrix}x_1\\ x_2\end{bmatrix}$ 值。编程实践中一般可以用迭代误差 eps 或者迭代次数 k 来控制迭代进程。

至此，基于上面的理论分析，我们就可以编写程序 16-7 来对某个具体的二元非线性方程组进行求解。

程序16-7　求解非线性方程组示例：二元二次方程

```
proc ds2;
  data dsZ / overwrite=yes;
    dcl double z;

    /*计算向量 F */
    method getVectorF(double x[2], double f[2,1]);
      f[1,1] = x[1]**2 - 10*x[1] + x[2]**2 + 8;
      f[2,1] = x[1]*x[2]**2 + x[1] - 10*x[2] + 8;
    end;

    /*计算雅可比矩阵 */
    method getMatrixJ(double x[2], double j[2,2]);
      j[1,1] = 2*x[1]-10;  j[1,2] = 2 * x[2];
      j[2,1] = x[2]**2+1; j[2,2] = x[1]*2*x[2]-10;
    end;

    /*返回矩阵的无穷范数，即取向量的最大值 */
    method norm(double x[*]) returns double;
      dcl int i;
      dcl double maxabsx;

      maxabsx abs(x[1]);
      do i 2 to dim(x);
        if abs(x[i])> maxabsx then maxabsx abs(x[i]);
      end;
```

```
    return maxabsx;
  end;

  /*方程求解主程序入口 Main*/
  method init();
    dcl package matrix m_jx;
    dcl double j[2,2];

    dcl package matrix m_fx;
    dcl double f[2,1];

    dcl package matrix m_y;
    dcl double y[2,1];

    dcl package matrix m_xk_n;
    dcl double x[2];

    dcl package matrix m_xk;
    dcl double xk[2];

    dcl double xk_diff[2];
    dcl package matrix m_ji;

    dcl int k  i;
    dcl double eps;

    m_jx = _new_ matrix(2, 2);
    m_fx = _new_ matrix(2, 1);
    m_xk = _new_ matrix(2, 1);

    eps = 1;
    k = 0;

    /* 假定 xk 是F(x)=0 第1次近似根 */
    xk[1] = 0 ;
    xk[2] = 0;
    put '>(' xk[1] best5.2',' xk[2] best5.2')';

    do while(eps >10**-9 and k < 1000 );
      getVectorF(xk, f);/* 利用 x[k] 计算函数向量 F */
      m_fx.load(f);

      getMatrixJ(xk, j);/* 利用 x[k] 计算雅可比矩阵 J */
      m_jx.load(j);

      m_ji = m_jx.inverse();/* 对雅可比矩阵求逆*/
      m_y = m_ji.mult(m_fx);/* 与函数向量相乘*/

      m_xk.load(xk);
      m_xk_n = m_xk.sub(m_y);/*计算下一逼近点*/
      m_xk_n.toarray(x);
      xk := x;

      m_y.toarray( xk_diff);/* 计算迭代误差 */
      eps = norm( xk_diff );

      k + 1;
      put k '(' x[1] best5.2 ',' x[2] best5.2 ')' eps= best9.7;
    end;
    put '! (' x[1] best5.2 ',' x[2] best5.2 ')' eps  best9.7;
    do i 1 to dim(x);
      z x[i]; output;
```

```
    end;
  end;
enddata;
run;
quit;
proc print data=dsZ;run;
```

系统日志输出如下，结果输出如图 16-3 所示。

```
> (        0,          0)
1 (      0.8,       0.88) eps=0.88
2 (    0.992,      0.992) eps=0.1917872
3 (        1,          1) eps=0.0082568
4 (        1,          1) eps=0.0000315
5 (        1,          1) eps=3.935E-10
! (        1,          1) eps=3.935E-10
```

即程序迭代 5 次求得方程组的根为 x_1=1，x_2=1，代入原方程组验证无误。

Obs	z
1	1
2	1

图 16-3　非线性方程组的根

对于其他多元非线性方程组，都可以用类似的方法进行求解。下面再举一个包含三角函数和指数函数的非线性方程组，并探讨如何用 SAS 编程进行求解。

$3x_1-\cos(x_2x_3)-0.5=0$

$x_1^2-81(x_2+0.1)^2+\sin(x_3)+1.06=0$

$e^{-x_1y}+20z+(10\pi-3)/3=0$

求解思路是一样的，首先可以定义 F(x) 为

$$f(x^k)=\begin{bmatrix} 3x_1-\cos(x_2x_3)-0.5 \\ x_1^2-81(x_2+0.1)^2+\sin(x_3)+1.06 \\ e^{-x_1x_2}+20x_3+(10\pi-3)/3 \end{bmatrix}$$

$F(x)$ 的 3 个分量 $f_1(x_1, x_2, x_3)$、$f_2(x_1, x_2, x_3)$，$f_3(x_1, x_2, x_3)$ 在 x^k 处分别对 x_1、x_2、x_3 求偏导数得到雅可比矩阵 $J(x^k)$，关于如何计算偏导数读者可查阅相关书籍和教材。

$$F'(x^k)\begin{bmatrix} \frac{\partial f_1(x^k)}{\partial x_1} & \frac{\partial f_1(x^k)}{\partial x_2} & \frac{\partial f_1(x^k)}{\partial x_n} \\ \frac{\partial f_2(x^k)}{\partial x_1} & \frac{\partial f_2(x^k)}{\partial x_2} & \frac{\partial f_2(x^k)}{\partial x_n} \\ \frac{\partial f_3(x^k)}{\partial x_1} & \frac{\partial f_3(x^k)}{\partial x_2} & \frac{\partial f_3(x^k)}{\partial x_n} \end{bmatrix} \begin{bmatrix} 3 & x_3\sin(x_2x_3) & x_2\sin(x_2x_3) \\ 2x_1 & 81\times 2(x_2+0.1) & \cos(x_3) \\ -x_2e^{-x_1x_2} & -x_1e^{-x_1x_2} & 20 \end{bmatrix}$$

即

$$J(x^k)\begin{bmatrix} 3 & x_3\sin(x_2x_3) & x_2\sin(x_2x_3) \\ 2x_1 & 81\times 2(x_2+0.1) & \cos(x_3) \\ x_2e^{x_1x_2} & x_1e^{x_1x_2} & 20 \end{bmatrix}$$

不同的非线性方程组求解就是其矩阵行列大小不同，初始向量 X_0、向量 F 和矩阵 J 的计算方式不同而已，通常要求具体问题具体分析。程序 16-8 展示了该三元非线性方程组的求解方法。

程序16-8 求解非线性方程组示例：包含三角函数和指数函数的三元二次方程

```
proc ds2;
  data dsZ / overwrite=yes; /*计算结果输出到数据集 dsZ 中*/
    dcl double z;/*输出结果的列名*/

    /*计算向量 F - 因方程组不同而不同 */
    method getVectorF(double x[3], double f[3,1]);
      f[1,1] = 3*x[1] - cos(x[2]*x[3]) - .5;
      f[2,1] = x[1]**2 - 81*(x[2]+.1)**2 + sin(x[3]) + 1.06;
      f[3,1] = exp(-x[1]*x[2]) + 20*x[3] + (10*constant('PI') - 3)/3.0;
    end;

    /*计算雅可比矩阵 J - 因方程组不同而不同  */
    method getMatrixJ(double x[3], double j[3,3]);
      j[1,1] = 3;
      j[1,2] = x[3]*sin(x[2]*x[3]);
      j[1,3] = x[2]*sin(x[2]*x[3]);
      j[2,1] = 2*x[1];
      j[2,2] = -162*(x[2]+.1);
      j[2,3] = cos(x[3]);
      j[3,1] = -x[2]*exp(-x[1]*x[2]);
      j[3,2] = -x[1]*exp(-x[1]*x[2]);
      j[3,3] = 20;
    end;

    /*返回矩阵的无穷范数，即取向量的最大值 */
    method norm(double x[*]) returns double;
      dcl int i;
      dcl double maxabsx;
      maxabsx=abs(x[1]);
      do i=2 to dim(x);
        if abs(x[i])> maxabsx then maxabsx=abs(x[i]);
      end;
      return maxabsx;
    end;

    /*非线性方程组求解主程序 Main*/
    method init();
      dcl package matrix m_jx;
      dcl double j[3,3];

      dcl package matrix m_fx;
      dcl double f[3,1];

      dcl package matrix m_y;
      dcl double y[3,1];

      dcl package matrix m_xk_n;
      dcl double x[3];

      dcl package matrix m_xk;
      dcl double xk[3];

      dcl double xk_diff[3];
      dcl package matrix m_ji;
```

```
        dcl int k;
        dcl double eps;
        dcl int i;

        /*初始化三个矩阵，分别用于存储雅可比矩阵J， X向量和函数F向量*/
        m_jx = _new_ matrix(3, 3);
        m_fx = _new_ matrix(3, 1);
        m_xk = _new_ matrix(3, 1);

        eps = 1;
        k = 0;

        /* 假定初始向量 */
        xk[1] = 1 ;
        xk[2] = 1;
        xk[3] = 1;
        put '> (' xk[1] best5.2 ',' xk[2] best5.2 ',' xk[3] best5.2')';

        do while(eps > 10**-9 AND k<1000);
          getVectorF(xk, f);/* 利用 x[k] 计算函数向量 F */
          m_fx.load(f);

          getMatrixJ(xk, j);/* 利用 x[k] 计算雅可比矩阵 J */
          m_jx.load(j);

          m_ji = m_jx.inverse();/* 对雅可比矩阵求逆*/
          m_y = m_ji.mult(m_fx);/* 与函数向量相乘*/

          m_xk.load(xk);
          m_xk_n = m_xk.sub(m_y);/*计算下一逼近点*/
          m_xk_n.toarray(x);
          xk := x;

          m_y.toarray( xk_diff);/* 利用矩阵无穷范数衡量迭代误差 */
          eps = norm( xk_diff );

          k + 1;
          put k '(' x[1] best5.2 ',' x[2] best5.2 ',' x[3] best5.2')'
            eps= best9.7;
        end;
        put '! (' x[1] best5.2 ',' x[2] best5.2 ',' x[3] best5.2')'
          eps= best9.7;

        do i=1 to 3;
          z=x[i]; output;/*输出结果向量*/
        end;
      end;
    enddata;
  run;
  quit;
  proc print data=ds2; run;
```

系统输出如图 16-4 所示，方程组求解系统迭代 8 次后得到方程组的近似根为 x_1=0.5, x_2=0, x_3=−0.52，感兴趣的读者可以代入原方程组进行验证。

```
> (     1,     1,     1)
1 (  0.92, 0.461,  -0.5) eps 1.5033876
2 ( 0.501, 0.187, -0.52) eps=0.4186867
3 ( 0.501, 0.061,  0.52) eps   0.12628
4 (   0.5, 0.012,  0.52) eps 0.0495363
5 (   0.5, 61E 5,  0.52) eps 0.0110115
```

```
6 (    0.5, 18E-7, -0.52) eps=0.0006038
7 (    0.5, 2E-11, -0.52) eps=1.8264E-6
8 (    0.5,    -0, -0.52) eps=1.671E-11
! (    0.5,    -0, -0.52) eps=1.671E-11
```

Obs	z
1	0.50000
2	-0.00000
3	-0.52360

图 16-4　非线性方程组的根

第17章

数据结构——图

在数学上，图是表示两事物之间关系的基本方法，它由一系列顶点以及连接这些顶点的边组成的，通常用一个顶点（Vertex）组成的集合 V 和一个边（Edge）组成的集合 E 共同表示，其中 V 的元素为 V_0，V_1，…，V_n，而 E 的元素为二元组（V_i，V_j）。如果图的边具有方向性，该图称为有向图，否则为无向图。如果每条边都关联两个不同的顶点，且不存在两条边它们所关联的顶点是相同的，则该图为简单图，否则为多重图。如果去掉某个边可导致整个图分裂成两个不连通的图，则称那个边为桥，如图 17-1 中的边（V_1，V_4）就是一个桥。

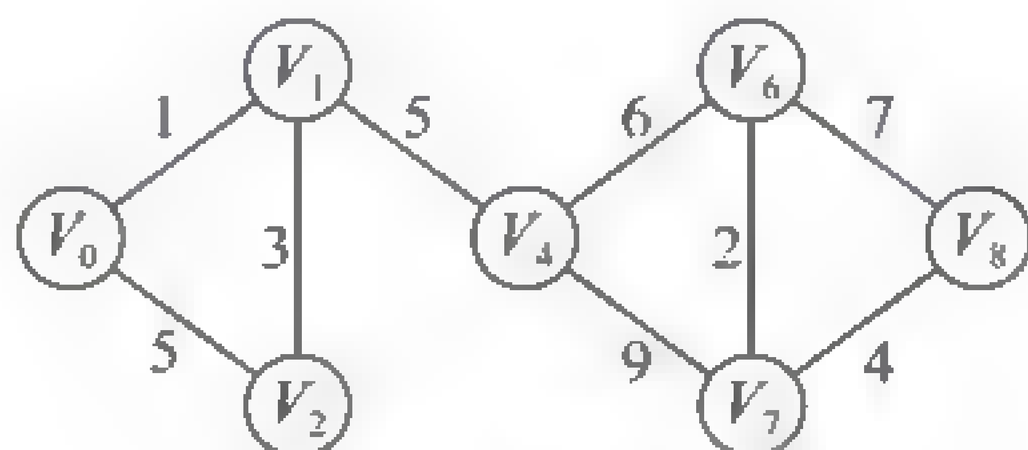


图 17-1　连通图与桥

在一个图中从顶点 V_i 到顶点 V_j 的一条路径是指从起点 V_i 到达终点 V_j 之间的顶点和边的集合，记为 V_i，V_k，…，V_j。如果顶点序列 V_i，V_k，…，V_j 没有任何相同（重复）的顶点，此路径（Path）称为简单路径。如果顶点 V_i 和顶点 V_j 不是某一条边的顶点，但它们之间存在通路，则该通路所包含的边长之和称为该路径的长度 d (Distance)。比如，图 17-2 中顶点 V_0 到顶点 V_3 的其中一条路径 V_0，V_1，V_3，其长度为 8，另一条路径 V_0，V_1，V_2，V_4，V_3 的长度为 7。

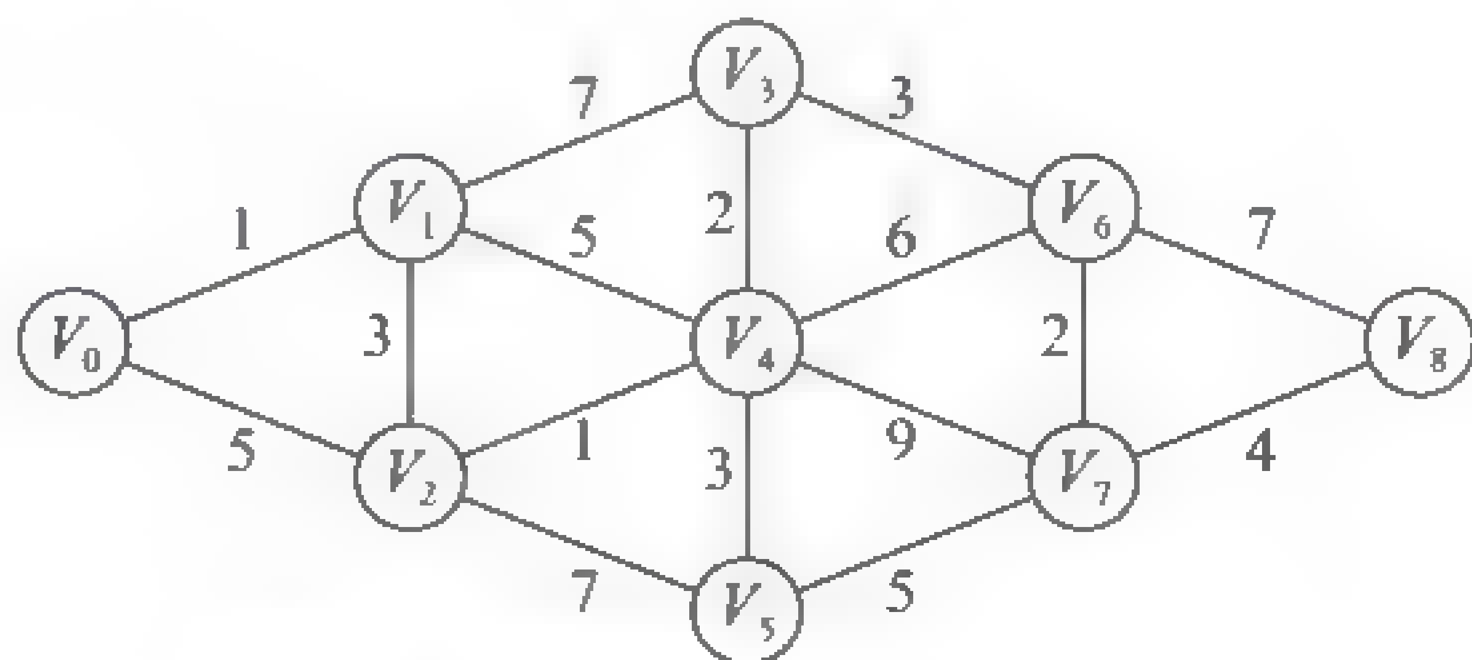


图 17-2　图与简单路径 V_0，V_1，V_3

对于有向图，如果某个顶点 V_i 有 d_i 条边以它为终点，d_0 条边以它为起点，则 d_i、d_0 分别称为该顶点的入度（In-degree）和出度（Out-degree），相应的边称为入边和出边。

在计算机系统中，图可由邻接表或邻接矩阵表示，其中图的邻接矩阵表示是唯一的。

邻接矩阵是一种特殊的二维数组，其行列下标分别表示邻接关系的起点和终点，数组的值表示边的长度或者其他含义。比如，$E[V_1, V_2]=3$ 表示从顶点 V_1 到顶点 V_2 之间有边相连，其长度为 3 。上面的无向图可用如下二元组 $G(V, E)$ 完整表示（见图 17-3）。

$$V=(V_0\quad V_1\quad V_2\quad V_3\quad V_4\quad V_5\quad V_6\quad V_7\quad V_8)$$

$$E=\begin{pmatrix} 0 & 1 & 5 & \infty & \infty & \infty & \infty & \infty & \infty \\ 1 & 0 & 3 & 7 & 5 & \infty & \infty & \infty & \infty \\ 5 & 3 & 0 & \infty & 1 & 7 & \infty & \infty & \infty \\ \infty & 7 & \infty & 0 & 2 & \infty & 3 & \infty & \infty \\ \infty & 5 & 1 & 2 & 0 & 3 & 6 & 9 & \infty \\ \infty & \infty & 7 & \infty & 3 & 0 & \infty & 5 & \infty \\ \infty & \infty & \infty & 3 & 6 & \infty & 0 & 2 & 7 \\ \infty & \infty & \infty & \infty & 9 & 5 & 2 & 0 & 4 \\ \infty & \infty & \infty & \infty & \infty & \infty & 7 & 4 & 0 \end{pmatrix}$$

图 17-3　图的二元组表示

17.1　深度优先和广度优先遍历

图作为比树更加复杂的数据结构，对图中每个顶点的遍历是首先需要考虑的编程问题，它也包括深度优先和广度优先两种遍历方法。

（1）深度优先遍历：其步骤为从图的某顶点 V_0 出发，访问 V_0；然后选择一个与 V_0 相邻且没有访问过的顶点 V_i 进行访问；再从 V_i 出发选择一个与 V_i 相邻且未被访问的顶点 V_j 进行访问，依次继续。如当前被访问过的顶点的所有邻接顶点都已经被访问过，则退回到已访问的顶点序列中最后一个还有未被访问的相邻顶点 V_k，从 V_k 出发按照同样的方法继续遍历，直到图中所有顶点都被访问为止。此方法与二叉树的深度优先遍历具有一定的相似性。比如图 17-2 按照深度优先遍历顺序进行遍历，其访问遵循从左到右的顺序（见程序 17-4）。

```
0 -> 1, 1 -> 2, 2 -> 4, 4 -> 3, 3 -> 6, 6 -> 7, 7 -> 5, 7 -> 8。
```

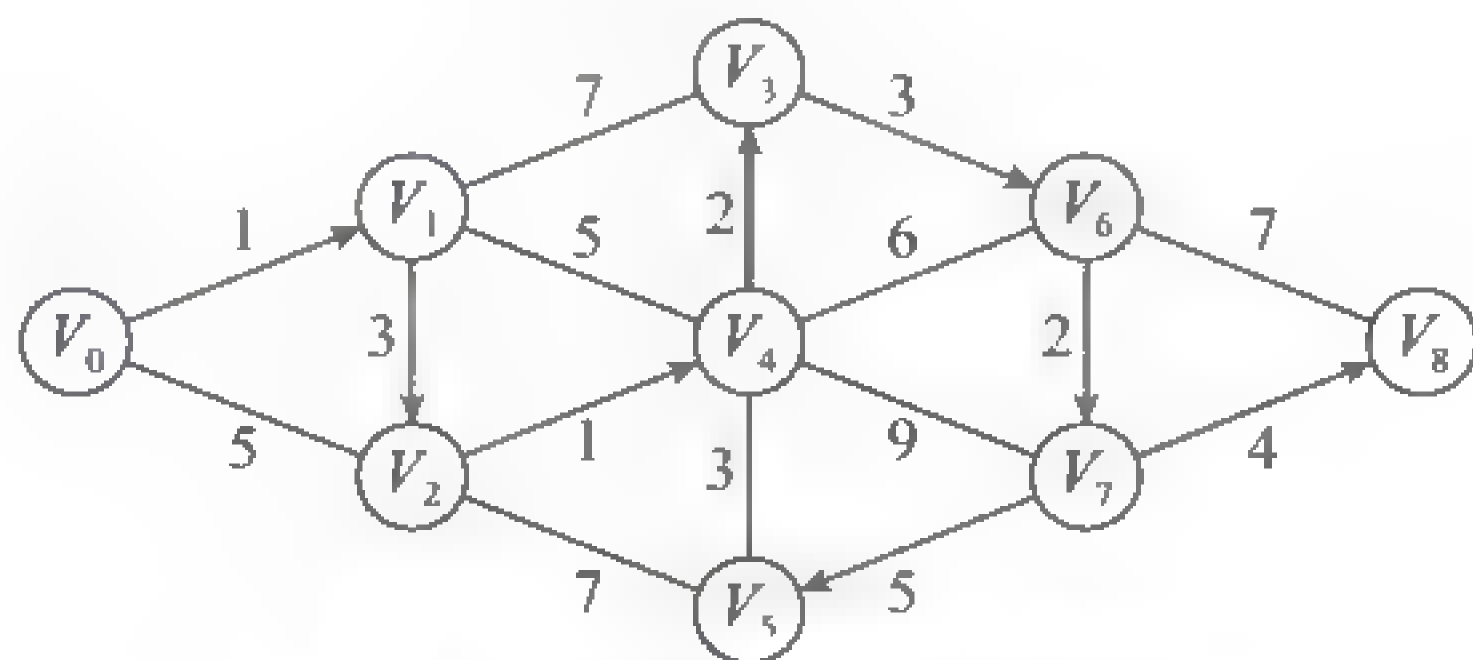


图 17-4　图的深度优先遍历

程序 17-1 是笔者实现深度优先遍历的算法，其参数 Vertex 和 Edge 分别为图的二元组 $G(V, E)$ 表示的顶点集和边集。

程序17-1　图的深度优先遍历算法实现

```
proc fcmp outlib=work.funcs.graph;

  /*图的深度优先遍历辅助函数*/
  function DFS (vertex[*] $, edge[*,*], visited[*], v);
    outargs visited;
    visited[v]=1;
    do i=1 to dim(vertex);
      if i^=v AND edge[v,i]^=constant("BIG") then do;
        if visited[i]^=1 then do;
          put (v-1) "->" (i-1); /* 遍历路径 */
          rc=DFS(vertex, edge, visited, i);
        end;
      end;
    end;
    return(1);
  endsub;

  /*图的深度优先遍历主函数*/
  function DFSTraverse (vertex[*] $, edge[*,*]);
    array visited[1] /nosymbols;
    call dynamic_array(visited, dim(vertex));
    do v=1 to dim(vertex);
      if visited[v]^=1 then rc=DFS(vertex, edge, visited, v);
    end;
    return (1);
  endsub;
run;
quit;
```

然后我们就可以在 DATA 步中使用该算法。首先要用二元组表示该图，共有 9 个顶点，16 条边。那些彼此不相连的顶点之间的路径长度被初始化为无穷大。为了让准备图的数据结构代码可复用，将它包装为 SAS 宏 *%PrepGrap*() 进行调用（见程序 17-2）。

程序17-2　图的深度优先遍历示例

```
%macro PrepGraph();/*准备图的数据结构*/
  numVertexes=9;
  numEdges=16;

  /*1. 生成图的顶点 V={ V0…V8 }, 下标基于1 */
  array vertex[9] $   _temporary_ ;
  do i=1 to 9;
    x=i-1;
    vertex[i]='V'|| trim(left(x));
  end;

  /*2.1 生成图的边 E={ numVertexes X numVertexes }, 初始化为无穷大*/
  INFINITY=CONSTANT('BIG');
  array edge[9,9] _temporary_;
  do i=1 to numVertexes;
    do j=1 to numVertexes;
      if i=j then edge[i,j]=0;
      else do;
        edge[i,j]=INFINITY;
        edge[j,i]=INFINITY;
      end;
    end;
  end;
  /*2.2 构造顶点之间的边, 下标基于1*/
```

```
    edge[1,2]=1;  edge[1,3]=5;
    edge[2,3]=3;  edge[2,4]=7;  edge[2,5]=5;
    edge[3,5]=1;  edge[3,6]=7;
    edge[4,5]=2;  edge[4,7]=3;
    edge[5,6]=3;  edge[5,7]=6;  edge[5,8]=9;
    edge[6,8]=5;
    edge[7,8]=2;  edge[7,9]=7;
    edge[8,9]=4;

    /*2.3 将上三角矩阵赋给下三角，图的数据结构构造完毕*/
    do i=1 to numVertexes;
      do j=i to numVertexes;
        edge[j,i]=edge[i,j];
      end;
    end;
%mend;

/*测试程序开始*/
options cmplib=work.funcs;
data _null_;
  %PrepGraph;
  put 'DFS Traverse';
  rc=DFSTraverse(vertex, edge); /*3. 图的深度优先遍历*/
run;
```

运行上面的代码后，系统输出如图 17-5 所示。



图 17-5　图的深度优先遍历

（2）广度优先遍历：其步骤为从图的某顶点 V_i 出发，访问 V_i；然后标记该顶点为已访问，接着访问 V_i 所有未被访问过的 K 个邻接点 V_{i1}，V_{i2}，…，V_{ik} 并将它们标记为已访问；然后再按照 V_{i1}，V_{i2}，…，V_{ik} 的顺序依次访问它们所有未被访问过的邻接点，并将它们标记为已访问；依此类推直到图中所有和初始点 V_i 连通的顶点被访问过为止。前面的图按广度优先顺序遍历为

```
0 -> 1, 0 -> 2; 1 -> 3, 1 -> 4, 2 -> 5; 3 -> 6, 4 -> 7; 6 -> 8。
```

遵循从左到右，从上到下的访问顺序（带箭头的边）（见图 17-6）。

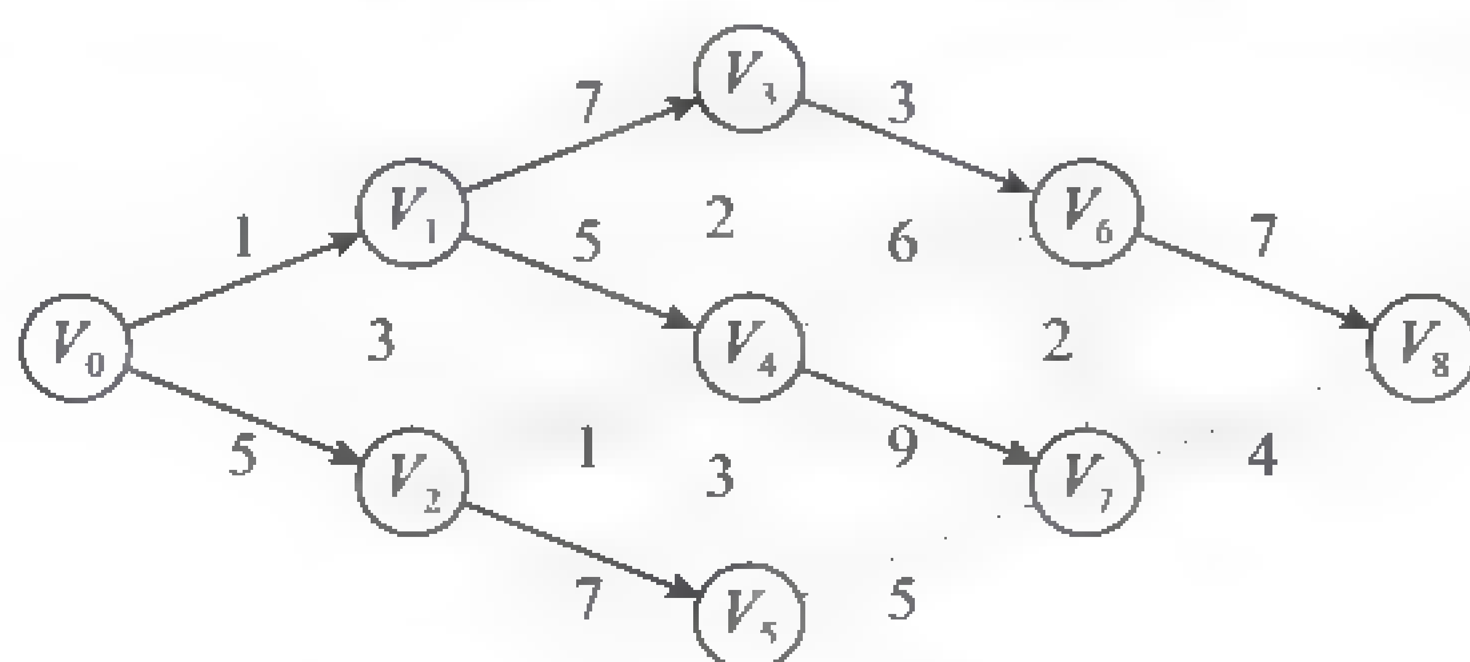


图 17-6　图的广度优先遍历

程序 17-3 是笔者用 SAS 实现的广度优先遍历算法，其中用了一个虚拟队列编程技巧实现动态添加下一次需要访问的节点集合。

程序17-3　图的广度优先遍历算法实现

```
proc fcmp outlib=work.funcs.graph;
  /*图的广度优先遍历*/
  function BFSTraverse (vertex[*] $, edge[*,*]);
    array visited[1] /nosymbols;
    call dynamic_array(visited, dim(vertex));

    array queue[1] /nosymbols;
    call dynamic_array(queue, dim(vertex));

    head=1;
    tail=1;

    do v=1 to dim(vertex);
      if visited[v]^=1 then do;
        visited[v]=1;

        queue[tail]=v;
        tail=tail+1;

        do while (head<tail);
          v2=queue[head];
          head=head+1;

          do i=1 to dim(vertex);
            if i^=v2 AND edge[v2,i]^=constant("BIG") then do;
              if visited[i]^=1 then do;
                put (v2-1) "->" (i-1);/*遍历路径 */
                visited[i]=1;

                queue[tail]=i;
                tail=tail+1;
              end;
            end;
          end;
        end;
      end;
    end;
    return (1);
  endsub;
run;
quit;
```

复用前面例子中所用的图的二元组图数据结构，调用广度优先遍历算法只需将调用方法名改为 BFSTraverse 即可（见程序 17-4）。

程序17-4　图的广度优先遍历示例

```
options cmplib=work.funcs;
data _null_;
  %PrepGraph;
  put 'BFS Traverse';
  rc=BFSTraverse(vertex, edge);  /*图的广度优先遍历*/
run;
```

此时系统输出结果如图 17-7 所示。

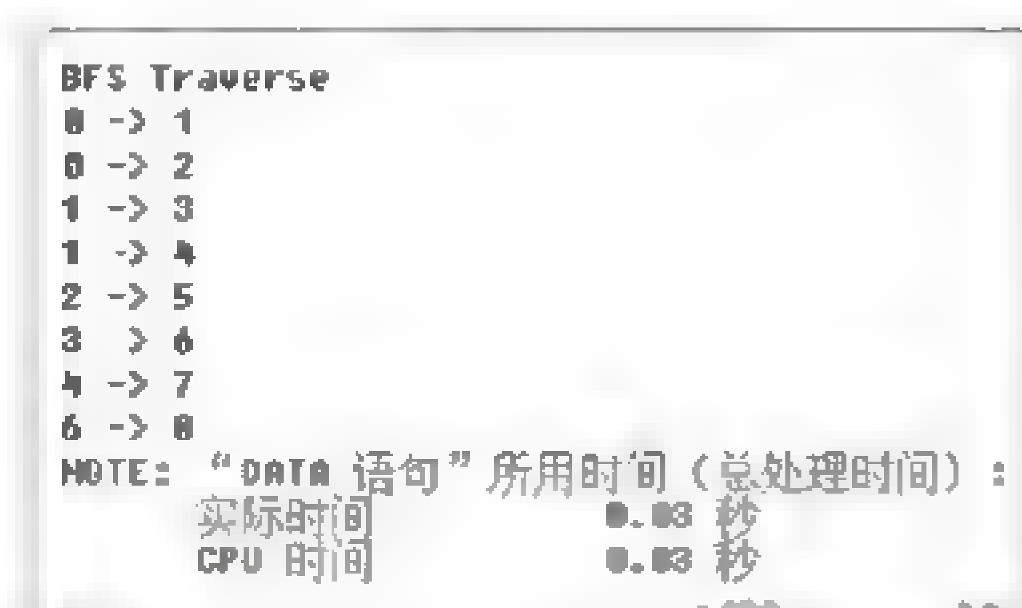


图 17-7　图的广度优先遍历

图在表述事物之间的特定关系方面非常有用，其中顶点表示事物，边表示事物之间的关系。1736 年欧拉关于柯尼斯堡七桥问题的论文，开启了图的研究而发展出数学分支“图论”。七桥问题说的是当时东普鲁士的柯尼斯堡市（见图 17-8 左），市区的河里有两座小岛，其中一座小岛与两岸各有 1 座桥相连，而另一座小岛与两岸各有 2 座桥相连，两座小岛之间还有 1 座桥相连。问题是我们能不能只走一遍就将所有的桥都走遍？



图 17-8　柯尼斯堡七桥拓扑图

如果将地图中陆地抽象为图的顶点，最终可以得到右侧的拓扑图（见图 17-8 右）。欧拉经过思考后把这类问题的本质归结为“一笔画”问题，即无重复一次遍历所有顶点的问题。他最后给出了这类问题的数学判定法则：**即所有顶点的边数为偶数，或者只有 2 个边数为奇数的顶点，可以无重复一次遍历；如果存在 2 个以上边数为奇数的顶点则不可能无重复一次遍历，它需要边数为奇数的顶点数的一半次数才能完全遍历。**

图在研究拓扑关系方面也具有重要意义，如著名的四色问题：是否任何一幅画在平面上的地图都可以仅用 4 种颜色染色，使任意两个相邻的国家颜色不同？只要将地图的接壤关系转化为以国家为顶点的无向图，从而变成探讨是否只用 4 种颜色给顶点进行染色而不出现邻接顶点同色的问题。

17.2　最短路径问题

在计算机软件工程领域研究最多的是图的最短路径问题，即如何在图中从顶点 V_i 到目标 V_j 点之间寻找一条最短路径，使从顶点 V_i 到顶点 V_j 的成本最小。现实生活中的汽车导航和路线规划问题，都是图的最短路径问题在实际生活中的应用，而最短路径问题又可分为以下 4 种情况。

（1）起始顶点确定，寻找它与目标顶点之间的最短路径问题。

（2）目标顶点确定，寻找从起始点到它之间的最短路径问题；在无向图中它与第 1 种情况等价，在有向图中可将所有路径方向反转变成第 1 类问题进行处理。

（3）给定两个点作为起点和目标顶点，求两者之间的最短路径，即路径规划问题。

（4）计算图中所有连通的顶点之间的最短路径，即全局最短路径计算问题。

各类最短路径问题可以分别采用 Dijkstra 算法、SPFA 算法和 Floyd-Warshall 算法进行求解，各种算法需要在实际应用中恰当选择。

17.2.1　Dijkstra算法

戴克斯特拉算法（Dijkstra's Algorithm）采用广度优先搜索来形成一个最短路径树，适用于不包含负权（即边长为负数）的有向图。它通过为每个顶点 v 都保留从给定起点 s 到它的最短路径 $d[v]$（若不连通的假定为无穷大）和最短路径前置顶点 $p[v]$；然后使用贪心算法从顶点 v 的所有下一个顶点集中找出最短路径对应的节点 w，其中 $d[w]=d[v]+edge(v,w)$。Dijkstra 算法时间复杂度为 $O(|V|)^2$，其中 $|V|$ 为顶点数如图 17-9 所示。

Dijkstra 算法的核心思想是深度上寻路：对于给定初始顶点 V_0，从与它连通且没有访问过的下一个节点中，找到最短的边，将该边的顶点 V 选入路径中，并保留 V_0~V 的最短距离；依此类推即可。对于具体的图（见图 17-9）其具体算法步骤如下所述。

$$\begin{pmatrix} 0 & 1 & 5 & \infty & \infty & \infty & \infty & \infty & \infty \\ 1 & 0 & 3 & 7 & 5 & \infty & \infty & \infty & \infty \\ 5 & 3 & 0 & \infty & 1 & 7 & \infty & \infty & \infty \\ \infty & 7 & \infty & 0 & 2 & \infty & 3 & \infty & \infty \\ \infty & 5 & 1 & 2 & 0 & 3 & 6 & 9 & \infty \\ \infty & \infty & 7 & \infty & 3 & 0 & \infty & 5 & \infty \\ \infty & \infty & \infty & 3 & 6 & \infty & 0 & 2 & 7 \\ \infty & \infty & \infty & \infty & 9 & 5 & 2 & 0 & 4 \\ \infty & \infty & \infty & \infty & \infty & \infty & 7 & 4 & 0 \end{pmatrix}$$

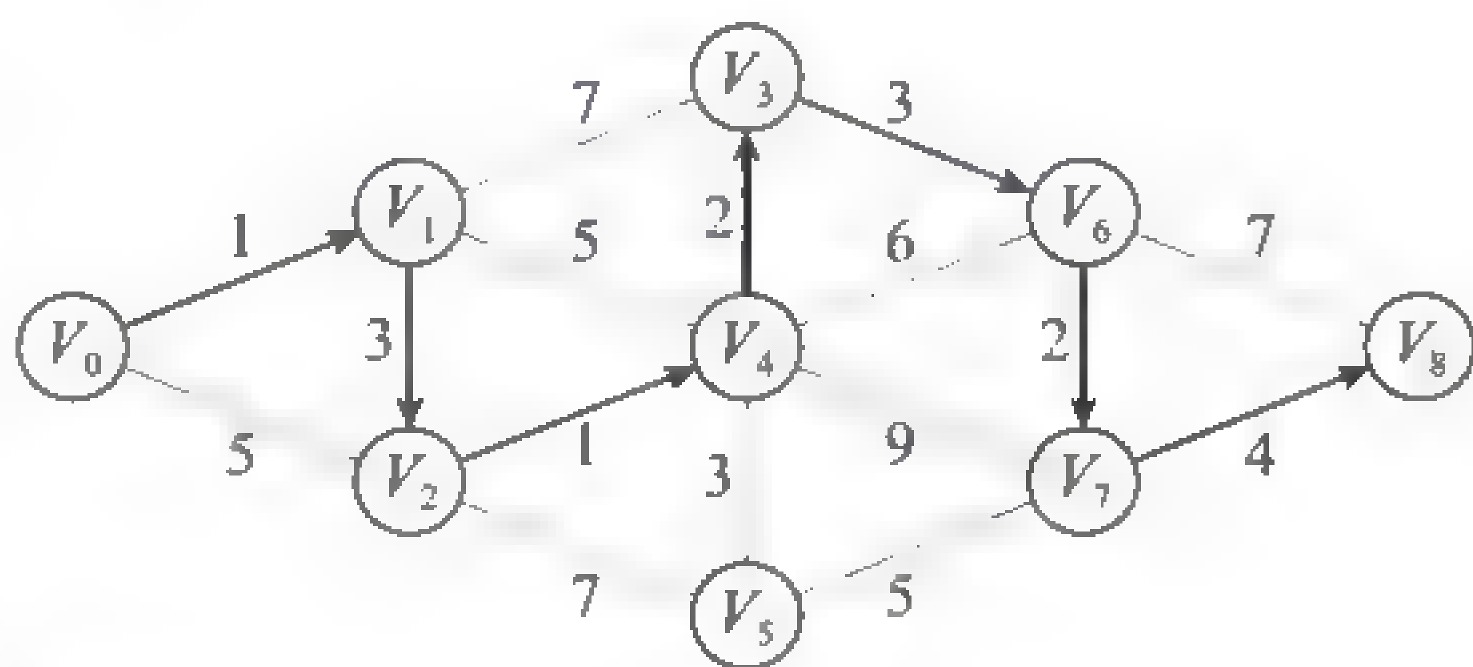


图 17-9　Dijkstra 算法范例

（1）对于给定起始顶点 V_0，我们知道 V_0 到它自身的距离为 0，即 $d[V_0]=0$。

（2）从 V_0 出发，考察 V_1，V_2，距离分别为 1, 5，择其短者 V_1，并保留最短距离 $d[V_1]=0+1=1$。

（3）从 V_1 出发，考察 V_2，V_3，V_4，距离分别为 3,7,5, 择其短者 V_2，保留最短距离 $d[V_2]=1+3=4$。

（4）从 V_2 出发，考察 V_4，V_5，距离分别为 1,7, 择其短者 V_4，保留最短距离 $d[V_4]=4+1=5$。

（5）从 V_4 出发，考察 V_3，V_5，V_6，V_7，距离分别为 2,3,6,9 择其短者 V_3，保留最短距离 $d[V_3]=5+2=7$。

（6）从 V_3 出发，考察 V_6，距离为 3，择其唯一 V_6，保留最短距离 $d[V_6]=7+3=10$。

（7）从 V_6 出发，考察 V_7，V_8，距离分别为 2，7 择其短者 V_7，保留最短距离 $d[V_7]=10+2=12$。

（8）从 V_7 出发，考察 V_5，V_8，距离分别为 5，4 择其短者 V_8，保留最短距离 $d[V_8]=12+4=16$。

（9）至此，V_0 到 V_8 的最短路径 16 已算出，路径为 V_0，V_1，V_2，V_4，V_3，V_6，V_7，V_8。

实际代码实现中，每个顶点 V 都保留有从起点 V_0 到它的最短距离和前置顶点信息，这样我们就可以获得从 V_0 开始到达任意一个连通顶点 V_i 的最短距离和路径信息。Dijkstra 算法要求距离必须是连续递增且该图具有全局最优解，它不支持负权处理。程序 17-5 为笔者用 SAS 语言实现的 Dijkstra 算法。

程序17-5 Dijkstra 单源最短路径算法，不处理负权

```
Proc fcmp outlib=work.funcs.graph;
  function shortestPath_Dijkstra( edge[*,*], v0, dist[*], prev[*]);
    outargs prev;
    outargs dist;

    vcount=dim(prev);

    array flag[1] /nosymbols;
    call dynamic_array(flag, vcount);

    do v=1 to vcount;
      flag[v]=0;           /*顶点 v 是否已寻得最短路径标志*/
      dist[v]=edge[v0 , v]; /*保留起点 v0 到顶点 v 的最短距离*/
      prev[v]=0;            /*顶点 v  的前置顶点*/
    end;
    dist[v0]=0;   /*v0 至 v0 距离为 0*/
    flag[v0]=1;   /*由于 v0 是起点，所以它不需寻找最短路径*/

    INFINITY=CONSTANT('BIG');
    /*开始主循环，每次求得v0到某个v顶点的最短路径*/
    do i=2 to vcount;
      /*搜索前置顶点中 d[u] 最小值的顶点 u*/
      dist_u=INFINITY;
      do v=1 to vcount;
        if (flag[v]^=1& dist[v]<dist_u) then do;
            dist_u=dist[v];
          u=v;
        end;
      end;
      flag[u]=1;
      /*搜索顶点 u 连通的边中最小的那个顶点 v，将其前置顶点置为u并修正最短距离*/
      do v=1 to vcount;
        if (flag[v]^=1 & (dist_u + edge[u , v]< dist[v])) then do;
          dist[v]=dist_u + edge[u , v];
          prev[v]=u;
        end;
      end;
    end;
    return (1);
  endsub;
run;
quit;
```

为了将最短路径信息打印出来，还需定义如下 printPath 函数，如程序 17-6 所示。

程序17-6 打印最短路径的辅助函数PrintPath 实现

```
proc fcmp outlib-work.funcs.graph;
  /*打印从 v0 出发到任意连通顶点的最短路径的长度和顶点信息*/
  function printPath(vertex[*] $, v0, dist[*], prev[*]);
    length path $ 32767;
    vcount=dim(dist);
    put "NOTE:源点到各点的最短长度和路径";
    do i=v0+1 to vcount;
      path= vertex[ i ];
      j=i;
      do while( prev[j]^=0);
        if prev[j]^=v0 then
          path=trim(left( vertex[ prev[j] ] )) || "->" ||
            trim(left(path));
        j=prev[j];
      end;
      path=trim(left( vertex[ v0 ] )) ||  "->" || trim(left(path));
    put  vertex[v0] "-> " vertex[i] ":" dist[i] 3." : " path  ;
  end;
  return (0);
  endsub;
run;
quit;
```

程序 17-7 是 Dijkstra 算法的测试程序，它用到了前面程序 17-2 中图的二元组数据结构。调用时只需要初始顶点 V_0、存放最短距离的数组 dist 和存放前置顶点信息的数组 prev 传入即可，计算结果如图 17-10 所示。

程序17-7 Dijkstra算法应用示例

```
options cmplib=work.funcs;
data _null_;
  %PrepGraph();
  v0=1;  /*源: V0=1*/
  array dist[9];
  array prev[9];
  rc=shortestPath_Dijkstra(edge, v0, dist, prev);
  rc=printPath(vertex, v0, dist, prev); /*打印最短路径信息*/
run;
```

```
V0 -> V1 :   1  : V0->V1
V0 -> V2 :   4  : V0->V1->V2
V0 -> V3 :   7  : V0->V1->V2->V4->V3
V0 -> V4 :   5  : V0->V1->V2->V4
V0 -> V5 :   8  : V0->V1->V2->V4->V5
V0 -> V6 :  10  : V0->V1->V2->V4->V3->V6
V0 -> V7 :  12  : V0->V1->V2->V4->V3->V6->V7
V0 -> V8 :  16  : V0->V1->V2->V4->V3->V6->V7->V8
NOTE: “DATA 语句”所用时间（总处理时间）:
      实际时间          0.05 秒
```

图 17-10 Dijkstra 算法输出

17.2.2 Bellman-Ford 算法

贝尔曼 - 福特算法（Bellman-Ford Algorithm）对 Dijkstra 算法的改进之处是它能够处理负权，也就是边的权值可以是负数的情况。它不再使用贪心算法选取尚未处理的具有最小权值的顶点，而是简单地对每一条边进行松弛操作，总共需要松弛操作的次数

为图顶点数 -1 次。该算法与 Dijkstra 算法在深度上寻路不同，它在广度上进行寻路，称为松弛操作。也就是说一开始每个边之间的估计值都比真实值大，而后通过反复计算，得到正确距离的边数不断增长直到所有的边都得到了正确路径。贝尔曼 - 福特算法需要较大量的运算，其时间复杂度为 $O(VE)$，实践中可使用基于 Bellman-Ford 算法的改进 Shortest Path Faster Algorithm（SPFA）算法。程序 17-8 为笔者用 SAS 实现的 Bellman-Ford 算法代码，适用于包含负权的单源最短路径查找。

程序17-8　Bellman-Ford算法实现，支持包含负权的单源最短路径查找

```
proc fcmp outlib=work.funcs.graph;
  /*最短路径：BellmanFord 算法，适用于单源最短路径，能处理负权*/
  function shortestPath_BellmanFord( edge[*,*], v0, dist[*], prev[*]);
    outargs dist;
    outargs prev;

    INFINITY=CONSTANT('BIG');
    /*初始化*/
    vcount=dim(dist );
    do i=1 to vcount;
      dist[i]=INFINITY;
      prev[i]=0;
    end;
    dist[v0]=0;

    /*重复对每一条边进行松弛操作*/
    do k=1 to vcount-1;
      do i=1 to vcount;
        do j=1 to vcount;
          if  edge[i,j]^=INFINITY then do;
            if dist[i]+ edge[i,j] < dist[j] then do;
              dist[j]=dist[i]+edge[i,j];
              prev[j]=i;
            end;
          end;
        end;
      end;
    end;

    /*检查负权环*/
    do i=1 to vcount;
      do j=1 to vcount;
        if  edge[i,j]^=INFINITY then do;
          if dist[i]+ edge[i,j] < dist[j] then do;
            put "ERROR: 图包含了负权环!";
            return(1);
          end;
        end;
      end;
    end;

    return(0);
  endsub;
run;
quit;
```

只需要将测试程序 17-7 代码中的 shortestPath_Dijkstra 改为 shortestPath_BellmanFord 即可输出从某个顶点出发的所有最短路径。

17.2.3　Floyd-Warshall 算法

弗洛伊德算法（Floyd-Warshall Algorithm）是任意两点之间最短路径的计算方法，它可以处理负权和有向图，但它也不能处理包含负权的回路。该算法的核心思想是基于动态规划。

假定 $D_{i,j,k}$ 为从顶点 i 到 j 的只以 $(1, 2, \cdots, k)$ 集合中的顶点为中间节点的最短路径长度。如果最短路径经过顶点 k，则 $D_{i,j,k}=D_{i,k,k-1}+D_{k,j,k-1}$；否则最短路径不经过顶点 k，则有 $D_{i,j,k}=D_{i,j,k-1}$。因此有 $D_{i,j,k}=\min(D_{i,j,k-1}, D_{i,k,k-1}+D_{k,j,k-1})$。该算法的时间复杂度为 $O(N^3)$，空间复杂度为 $O(N^2)$。

程序 17-9 是笔者用 SAS 语言实现的 Floyd-Warshall 算法，其中 dist[i,j] 表示从顶点 i 到顶点 j 的距离或代价，其取值为 INFINITY 表示两顶点之间没有通路。

程序17-9　Floyd-Warshall算法实现

```
proc fcmp outlib=work.funcs.math;
  /*最短路径3: Floyd 算法，任一对顶点间的最短路径*/
  function shortestPath_FloydWarshall( edge[*,*],  dist[*,*], path[*,*] $);
    outargs dist;
    outargs path;

    vcount=dim(dist, 1);

    INFINITY=CONSTANT('BIG');
    /*初始化*/
    do i=1 to vcount;
      dist[i,i]=0;
    end;

    do i=1 to vcount;
      do j=1 to vcount;
        dist[i,j]=edge[i,j];
        if dist[i,j]^=INFINITY then path[i,j]="V" ||compress((i-1) ||"->V" || compress(j-1));
        else path[i,j]="*";
      end;
    end;

    do k=1 to vcount;
      do i=1 to vcount;
        do j=1 to vcount;
          if dist[i,j]^=INFINITY & dist[i,k]^=INFINITY &
            dist[k,j]^=INFINITY then do;
            if dist[i,j] > dist[i,k]+ dist[k,j] then do;
              dist[i,j]=dist[i,k] + dist[k,j];
              path[i,j]=trim(left( path[i,k]))  || " " ||  trim(left( path[k,j] ));
            end;
          end;
        end;
      end;
    end;
    return(0);
  endsub;
run;
quit;
```

由于该算法是处理整个图所有顶点之间的最短路径，其输出为距离矩阵和路径矩阵，因此，需要定义两个专门用于输出距离和路径的函数 print_dist 和 print_path（见程序 17-10）。

程序17-10 输出距离矩阵和路径矩阵的辅助函数

```
proc fcmp outlib work.funcs.math;
  function print dist( dist[*,*]);
    length line $ 32767;
    vcount=dim(dist, 1);

    do i=1 to vcount;
      line="";
      do j=1 to vcount;
        if j>1 then line= trim(line) |  " ,";

        if dist[i,j]=CONSTANT('BIG') then do;
          line=  trim(line) || "*";
        end;
        else do;
          line=  trim(line) || trim(left( put(dist[i,j],4.0) ));
        end;
      end;
      put line;
    end;
    return(0);
  endsub;

  function print_path( path[*,*] $);
    length line $ 32767;
    vcount=dim(path, 1);

    do i=1 to vcount;
      line="";
      do j=1 to vcount;
        if j>1 then line= trim(line) || ",";
        line=  trim(line) || put(path[i,j], $6.);
      end;
      put line;
    end;
    return(0);
  endsub;
run;
quit;
```

程序 17-11 演示了如何用 Floyd-Warshall 算法对前面的数据进行计算，结果如图 17-11 所示。纵坐标 / 横坐标分别反映源点到目标点之间的最短路径，以及源点到目标点的路径信息。比如，第 1 行第 3 列为 4，表示图中顶点 V_0 到 V_2 的最短路径为 4, 其路径序列为 V_0->V_1 V_1->V_2。

程序17-11 Floyd-Warshall 算法应用示例

```
data _null_;
  %PrepGraph();

  array dist[9,9];
  array path[9,9] $  32767;
  rc=shortestPath_FloydWarshall(edge, dist, path);

  put "NOTE: 源点到各点的最短路径长度";
  rc print dist(dist);

  put "NOTE: 源点到各点的最短路径";
  rc print path(path);
run;
```

```
NOTE: 源点到各点的最短路径长度
0 ,1 ,4 ,* ,* ,* ,* ,* ,*
1 ,0 ,3 ,6 ,4 ,* ,* ,* ,*
4 ,3 ,0 ,* ,1 ,4 ,* ,* ,*
* ,6 ,* ,0 ,2 ,* ,3 ,* ,*
* ,4 ,1 ,2 ,0 ,3 ,5 ,7 ,*
* ,* ,4 ,* ,3 ,0 ,* ,5 ,*
* ,* ,* ,3 ,5 ,* ,0 ,2 ,6
* ,* ,* ,* ,7 ,5 ,2 ,0 ,4
* ,* ,* ,* ,* ,* ,6 ,4 ,0
NOTE: 源点到各点的最短路径
V0 >V0,V0 >V1,V0 >V1 V1->V2,*,*,*,*,*,*
V1->V0,V1->V1,V1->V2,V1->V2 V2->V4 V4->V3,V1->V2 V2->V4,*,*,*,*
V2->V1 V1->V0,V2->V1,V2->V2,*,V2->V4,V2->V4 V4->V5,*,*,*
*,V3->V4 V4->V2 V2->V1,*,V3->V3,V3->V4,*,V3->V6,*,*
*,V4->V2 V2->V1,V4->V2,V4->V3,V4->V4,V4->V5,V4->V3 V3->V6,V4->V3 V3->V6 V6->V7,*
*,*,V5 >V4 V4 >V2,*,V5 >V4,V5 >V5,*,V5 >V7,*
*,*,*,V6->V3,V6->V3 V3->V4,*,V6->V6,V6->V7,V6->V7 V7->V8
*,*,*,*,V7->V6 V6->V3 V3->V4,V7->V5,V7->V6,V7->V7,V7->V8
*,*,*,*,*,*,V8->V7 V7->V6,V8->V7,V8->V8
NOTE: "DATA 语句"所用时间(总处理时间):
      实际时间            0.06 秒
      CPU 时间            0.07 秒
```

图 17-11　Floyd-Warshall 算法输出

本章主要讲述了图的基本知识，包括两类遍历算法和三种最短路径算法。读者可以参考笔者的 SAS 代码实现去开发用到图这种数据结构的分析应用。比如，在社交网络分析和链接分析中，图就是一种最基本的常用数据表达形式。

SAS 对图论、组合优化和网络分析等领域提供专业的分析包 SAS/OR（需要专门的软件授权），它可对实体（也叫顶点或节点）之间的关系（也叫边或联接）进行各种分析。其中 PROC OPTNET 和 PROC OPTGRAPH 分别支持 11 种和 17 种算法；用户也可使用 PROC OPTMODEL 网络求解器进行网络分析。SAS 过程步可使数据分析人员专注于数据准备和结果解读，而不是具体算法实现。程序 17-12 演示了如何使用 SAS/OR 进行最短路径分析，指定起点和终点的最短路径输出与前面 Dijkstra 算法输出相同（见图 17-12）。

程序17-12　SAS/OR 最短路径分析示例代码

```
/* 构造数据集: from 起点 to 终点 weight 权重*/
data mygraph;
  input from $ to $ weight @@;
  datalines;
v0 v1 1 v0 v2 5
v1 v2 3 v1 v3 7 v1 v4 5
v2 v4 1 v2 v5 7
v3 v4 2 v3 v6 3
v4 v5 3 v4 v6 6
v5 v7 5
v6 v7 2 v6 v8 7
v7 v8 4
;
/* 1. 指定起点和终点的最短路径*/
proc optgraph data_links=mygraph;
  shortpath out_paths=mypath source=" v0"  sink=" v8" ;
run;
proc print data mypath;
run;
/* 2. 处理包含负权时的最短路径*/
proc optgraph data links mygraph direction = directed;
  shortpath out paths mypath source " v0"  sink " v8" ;
run;
```

```
/* 3. 寻找所有节点之间的最短路径*/
proc optgraph data links=mygraph;
  shortpath out paths = shortpath paths;
run;
```

Obs	source	sink	order	from	to	weight
1	v0	v8	1	v0	v1	1
2	v0	v8	2	v1	v2	3
3	v0	v8	3	v2	v4	1
4	v0	v8	4	v4	v3	2
5	v0	v8	5	v3	v6	3
6	v0	v8	6	v6	v7	2
7	v0	v8	7	v7	v8	4

图 17-12　SAS/OR 最短路径分析示例输出

掌握了 SAS 语言以及利用它去开发数据结构和算法的基础后，我们就掌握了用 SAS 语言开发各种分析应用的编程技术。然而这一切还只能让我们停留在程序员的层次上。作为数据科学家，还需要掌握作为数据分析基础的统计学知识和数学知识，尤其是统计学有关的各种定律和定理，从基本的数据统计量和统计分布出发，掌握单变量数据分析技术，各种多变量数值分析方法，融会贯通数据分析的各种手段，最后才能"系统化"地掌握数据分析的精髓和终极目标：**基于历史数据构建有用的分析模型并从中获得洞见，实现对未知数据的分类和预测，从而获得 DIKW 模型中能够改变未来的所谓准则（Principal），达到"智慧"之级别。从分析的角度看，智慧是能够更好面对"持续变化且充满不确定性"的未来，指导当下行动以塑造尚未发生的未来，获取最优预期的终极知识。**

下　卷

统计学基础

统计学是使用科学方法收集、分析、呈现和解释数据的科学，它是数据分析科学的基础，主要由描述性统计和推断性统计构成。描述性统计是一大类统计的总称，主要是用于描述或总结所观察到数据的基本情况，目的是获得对数据的总体感觉以及评估数据质量。它的主要手段包括两大类：①用数据特征度量来反映数据的集中趋势、离散趋势、分布形状；②用表格工具或图形化方法来了解样本数据总体分布情况。

（1）前者运用的工具是反映数据特征的数字工具，包括均值、众数、中位数，四分位数、百分位数、方差、标准差、变异系数等反映数据集中和离散程度的一系列特征度量。对于某个随机变量，由于观测到的变量数据受到诸多因素影响，当其中任何一个因素都不起决定性作用时，其观测值会服从或近似服从正态分布，为衡量观测到的数据与正态分布的偏离情况，统计学中还引入偏度系数和峰度系数两个基础统计量。

（2）后者采用表格工具或图形化方法来了解数据的总体分布情况，包括频数分布表、直方图、饼图、散点图、盒形图等各种数据可视化手段。直方图是为了解在特定取值区间上，观测值出现的频次而引入的图形工具；盒形图则可以直观地反映四分位数和中位数，最小值和最大值以及离群值的分布情况，为了解数据的分布特征提供很好的直观工具。

18.1 数据特征度量

人类认识事物是通过事物的不同属性或特征去认识的，根据不同方面的属性和特征，人们会建立不同的指标去观测它，来获得描述事物的数据。数据分析的对象为多次观测事物所得到的指标数据，而分析数据的终极目的是了解事物本身。在一次观测中得到的数据可能因为随机误差或者其他原因而不能很好地反映事物本身，因此人们通过反复观测来获得更全面的认识。然而，再多次数的观测也不能代表事物本身，也就是说人类不能获得观测对象所有可能的全面认识。因此，就需要对有限的观测数据进行分析，用科学方法推断总体的情况，尽可能从有限数据中获得对事物的确定性认识。

在统计学上，待研究的对象的全体称为总体（Population），而从总体中抽取的一部分个体组成的集合称为样本（Sample），样本中每一个待考察的对象称为个体（Individual），而样本中个体的数目称为样本容量。假如图 18-1 中所有的点就是我们需要认识的事物，称为总体，每一个小方框内的点组成的集合就是一个样本，而每一个点就对应一个个体。

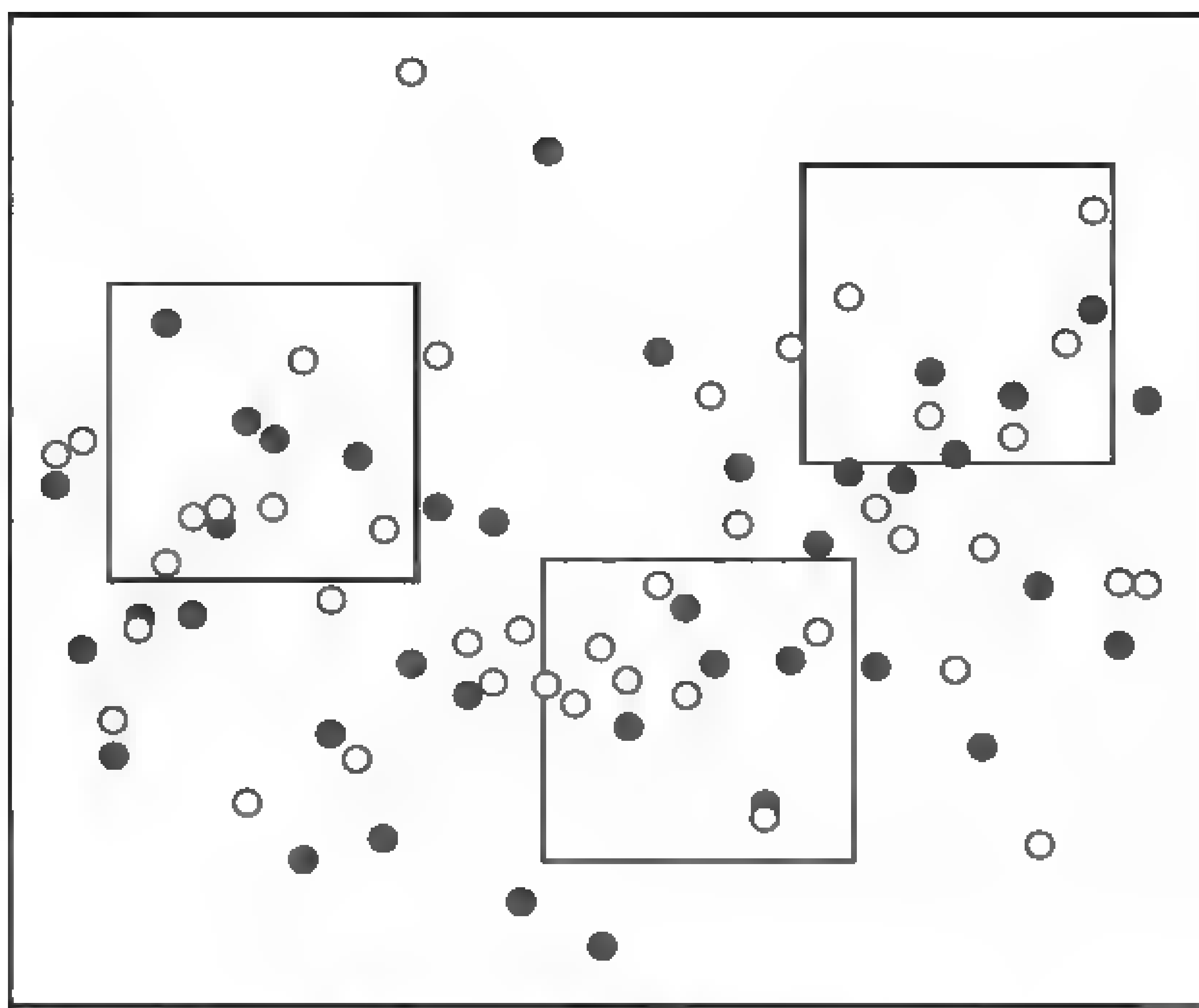


图 18-1　总体、样本与个体

用于描述事物特征或属性的单个变量 x，假定通过 n 次观测获得了 n 个观测结果 x_1，x_2，…，x_n，则这 n 个观测值就构成我们待研究的样本，其中 n 称为样本容量。为多方面观察事物，我们一般会从多个角度来观察，因此我们在单次观测中就可以得到多个变量的观测值。如果将观测到的数据用一个表格进行表示，则表格中每一行数据就对应统计学上的一次观测的结果，表格中的每一行在统计学上被称之为观测（Observation），而表格中每一列数据则代表一个观测指标，称为变量（Variable）。

对单个变量的数据进行分析时，需要了解其数据取值的特征，包括数据的集中趋势、离散趋势和分布形状等信息。比如，扔一个骰子获得点数，每一次都能够获得 6 个面中的一个，因此可能的取值就是 1~6，共 6 种情况，如果把抛骰子观测出现的点数当作一次试验，则该随机试验的样本空间为 {1,2,3,4,5,6}。现在假如张三抛了 3 次骰子（做了 3 次随机试验），获得了一个样本为 {1,5,3}；李四抛了 5 次骰子（做了 5 次随机试验），获得了另一个样本为 {2,3,1,4,5}。现在我们要研究抛骰子出现点数的规律，而我们又不可能做无限次试验，这就要求我们从这些有限样本数据中建立对抛骰子出现点数规律的“总体”的认识。

为了衡量样本数据的特征，在统计学上构造了一系列描述数据特征的概括值，称为样本统计量（Statistics）。一般以拉丁字母表示，如样本均值 $\bar{x}$、样本方差 S^2 等。很多时候，样本统计量被简称为统计量，它是基于样本数据计算得到的。下一小节将对各种样本统计量进行展开阐述。

然而，为了区分样本的数据特征与总体的数据特征，我们把描述总体特征，与样本统计量对应的概括值称为总体参数（Parameter），常被简称为参数，一般用希腊字母表示。比如，总体均值 μ、总体方差 σ^2 等，总体均值又称为期望（Expectation）。总体参数通常是未知数，由于我们不可能够获得“所有样本”进行计算，因此需要科学方法来对有限样本数据进行计算，并估计总体参数。

18.1.1 集中趋势度量

（1）均值（Mean，也称算术平均值）是样本中 n 个观测值 $x_1, x_2, \cdots x_n$，的算术平均值，它反映样本观测值的总和均匀分配在每个个体上的多少。其计算公式为观测数据的总和除以样本容量：

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

均值表示数据集中的趋势，均值越大，表示数据普遍取值较大，也就说数据有在取值较大区域集中的趋势。均值代表了观测的平均水平，是数据取值的中心位置。

然而均值这一统计量容易受极端个体的影响而掩盖问题，也就是说大多数个体很容易“被平均”。当数据分布并非正态分布时，均值不太适合作为数据集中趋势的度量。严谨的统计学方法需要客观看待每一个样本统计量优缺点，合理理解各统计量的描述价值。有打油诗为证：张村有个张千万，隔壁九个穷光蛋，平均起来算一算，人人都是张百万　然而理性告诉我们，“人人都是张百万”的说法对于张村大部分穷光蛋们而言是荒谬而可笑的。

另外，除了前面最常用的算术平均值，还有一些别的形式的平均值，可分别用于不同的计算场景。下面列出了各种形式的平均值计算公式以及它们的加权形式：

- 算术平均值：$A = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{1}{n}\left(x_1 + x_2 + \cdots x_n\right)$，其加权形式为 $\bar{x} = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}$
- 几何平均值：$G = \sqrt[n]{\prod_{i=1}^{n} x_i} = \sqrt[n]{x_1 x_2 \cdots x_n}$
- 调和平均值：$H = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}} = \frac{n}{\left(\frac{1}{x_1} + \frac{1}{x_2} + \cdots + \frac{1}{x_n}\right)}$，其加权形式为 $\bar{x} = \frac{\sum_{i=1}^{n} w_i}{\sum_{i=1}^{n} \frac{w_i}{x_i}}$
- 平方平均值：$M = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2} = \sqrt{\frac{1}{n}\left(x_1^2 + x_2^2 + \cdots + x_n^2\right)}$

（2）众数（Mode）就是样本中出现次数最多的那个观测值，它在一定程度上也可以用描述样本数据的集中程度。其数学表达形式为

$$\text{Mode} = x_i \ \text{ if } \ \text{Freq}(x_i) = \max$$

众数代表观测值的一般水平，就是最常出现的那个观测值。统计学上样本的众数可以有1个或多个，也可以没有。当所有的观测值出现次数一样多时，众数就不存在。而当样本中观测值出现次数并列最多时，则这些观测值并列为众数，这时众数就有多个。

由于众数是统计数据出现的次数，而跟数据本身的类型和取值大小无关，因此众数在无法计算均值时非常有用，如待分析的数据只有定性信息，它只包含描述性信息或者顺序性信息。

众数在统计分布曲线中是概率密度函数峰值所对应的横坐标 x 值，也是频数分布直方图中最高的那个柱子对应的横坐标 x 值。众数的好处是不受极端值（极大或极小）的影响。

比如，前面张村人民富裕程度的描述就变为：张村有个张千万，隔壁九个穷光蛋，若按多数算一算，大家都是穷光蛋。不管如何，张村人民大多数人确实是穷光蛋，这种认识还是比“人人都是张百万”更加准确的。由此看来，众数的确能较好地反映数据的集中趋势，但另外由于众数的大小只跟经常出现的那部分数据有关，它仍然有以偏概全的风险。

（3）中位数（Median）：既然均值和众数都各有优缺点，那么是否有更好反映数据集中程度的特征度量呢？如果我们将样本数据按照数值大小顺序排列，取位于最中间的那个数或者最中间那两个数的平均值，是否它应该比平均值和众数在反映数据集中趋势上更加优越？统计学上这个数被称为中位数，就是中间位置的那个数。中位数能确保大于它和小于它的样本个数是一样多的，故又称中值。

它的数学表达形式为

$$\text{Median}=x_{(n+1)/2} \quad \text{当 } n \text{ 为奇数}$$
$$\text{Median}=(x_{n/2}+x_{n/2+1})/2 \quad \text{当 } n \text{ 为偶数}$$

即当样本容量 n 为奇数，中位数为位于最中间的那个数值，即排序后位于第 $(n+1)/2$ 个样本的值，否则中位数为第 $n/2$ 个与第 $n/2+1$ 个样本之和的平均值。由于中位数是计算所得，中位数可能是样本中一个实际存在的样本值，也可能是样本中根本不存在的一个数值。中位数代表数据总体的中间水平，在统计分布曲线中它对应的 x 值刚好将概率密度曲线下方的面积分成左右相等的两部分（见图 18-2）。

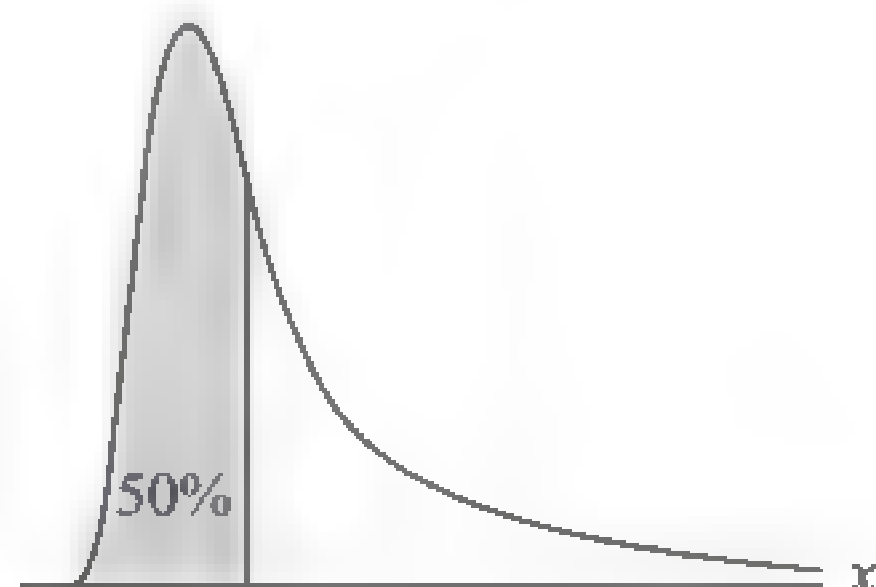


图 18-2　统计分布曲线与中位数

中位数既降低了均值引发的“被平均”风险，又减少了众数以偏概全的风险。极端值和部分数据的改变对中位数没有影响，该样本统计量具有较强的稳健性。正如打油诗所说：张村有个张千万，隔壁九个穷光蛋，若按中间算一算，最坦然是穷光蛋。中位数可保证比它大和比它小的观测个数是一样多的，也就说比中位数富裕的和比它穷困的各占一半，因此中位数收入的人会因为比上不足比下有余而更加坦然。

从中位数往更一般地推论，统计学上引入四分位数（Quartile）和百分位数（Percentile）等统计量用于描述数据分布的位置。其中四分位数是将观测值从小往大排列，然后将出现的频数划分为 4 个相等的部分，则从小到大排列占比为 25% 所对应的变量值称为下四分位数 Q_1，50% 为中四分位数 Q_2（即中位数），75% 为上四分位数 Q_3，而 Q_4 就是最大值（见图 18-3）。概率分布曲线下方的面积，就是观测值的数据出现的概率，最左侧占总体面积 25% 所对应的数值就是 1/4 分位数 Q_1、3/4 分位数为左侧 75% 面积右侧边界所对应的分位数 Q_3，中位数 Q_2 就是将曲线下方面积分为左右均等的那个 x 值。

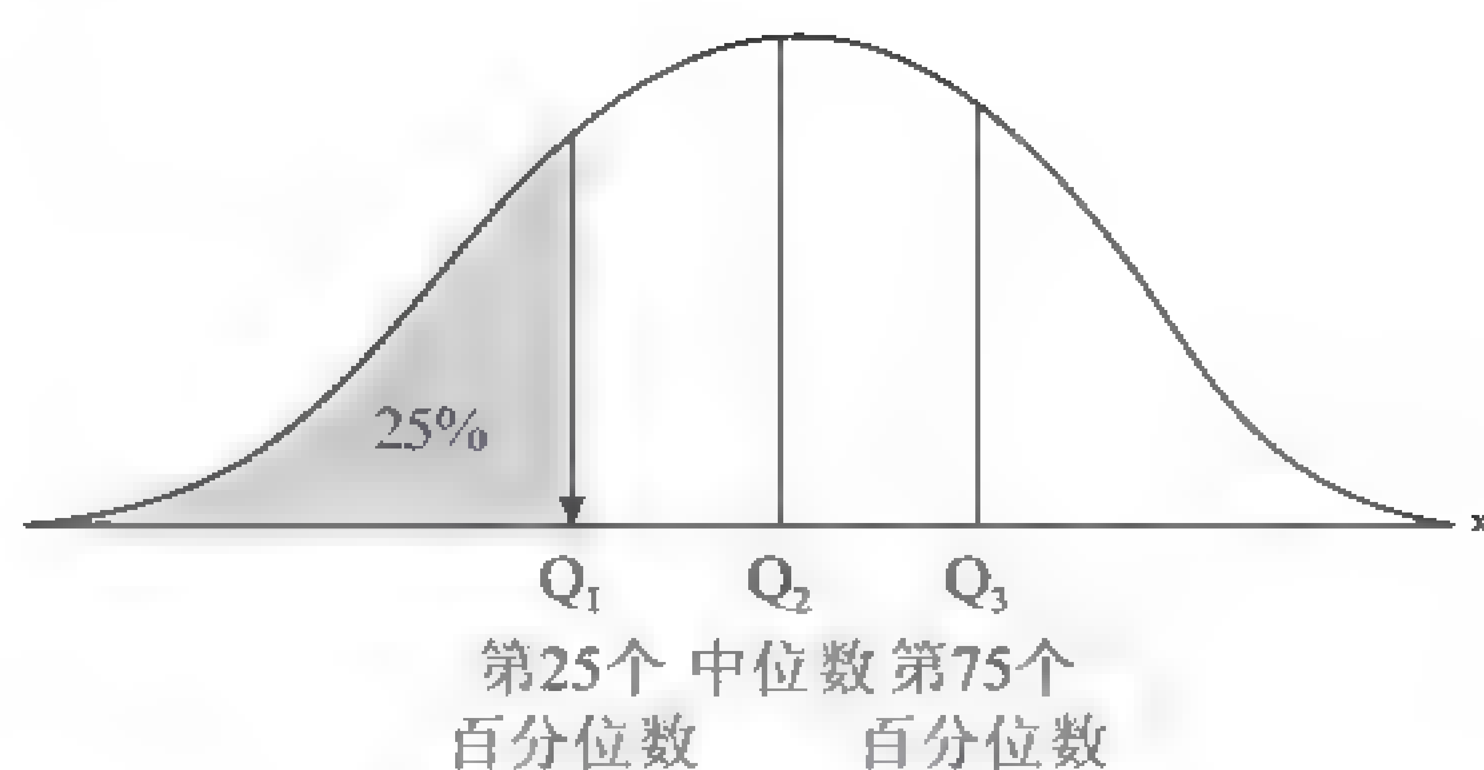


图 18-3　分布相关的四分位数

更一般地，可以构造出一系列的百分位数，常用的为两端占比为 1%、5% 和 10% 的那几个百分位数，分别记为百分位数 P_1、P_5、P_{10}、P_{90}、P_{95}、P_{99}（见图 18-4）。

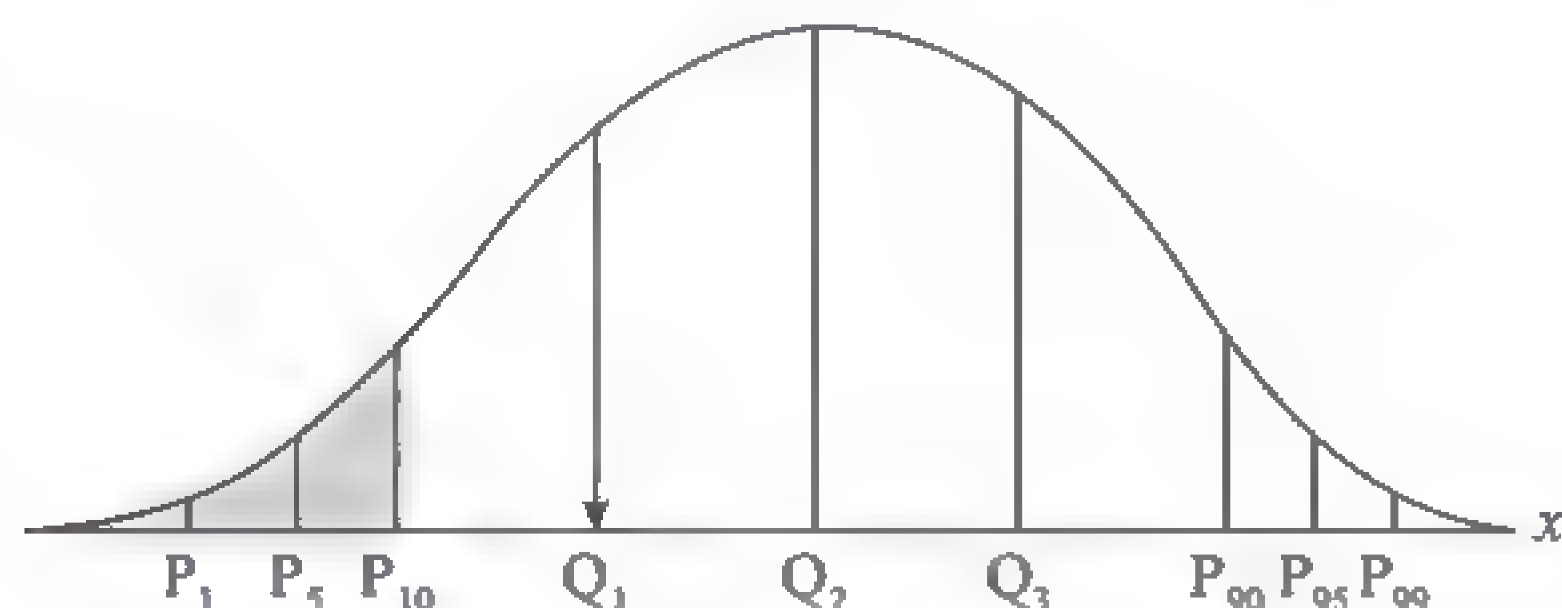


图 18-4　四分位数与百分位数

实践中百分位数的计算方法比中位数复杂。主要有如下两种计算方式，其中方法 2 是 SAS 等专业统计软件内部使用的百分位数计算方法，更加精确一点。假如计算第 25% 百分位数，则下文中的 p 值为 0.25。

方法 1：首先需要将 n 个观测值从小到大排列，假定 x_j 为排序后第 j 个数；计算数值 $A=n\times p$，把 A 的整数部分记为 i，小数部分记为 f。如果 A 刚好是整数，则第 p 百分位数为 $(x_i+x_{i+1})/2$，否则为 x_{i+1}。

方法 2：首先需要将 n 个观测值从小到大排列，假定 x_j 为排序后第 j 个数；计算数值 $A=(n+1)\times p$，把 A 整数部分记为 i，小数部分记为 f。如果 A 刚好是整数，则第 p 百分位数为 x_i，否则为 $(1-f)\times x_i+f\times x_{i+1}$。

18.1.2　离散趋势度量

为衡量数据的离散程度，统计学中引入极差、半极差、方差、变异系数、离差平方和等样本统计量，描述样本数据在分布空间上的离散程度。

（1）极差（Range）就是样本数据在分布空间上的跨度。最小观测值（min）和最大观测值（max）分别代表样本中两个方向的极端情况，而它们相减的差就可表示观测数据的波动范围，所以极差又称全距。其数学表达形式为

$$\text{Range}=\max(x_i)-\min(x_i)$$

同理，也可以定义上下四分位数的差，称为半极差（Interquartile Range，QRANGE），

也称四分位极差或四分位间距。半极差衡量的是中间 50% 观测值的波动范围。其数学表达形式为

$$QRANGE=Q_3-Q_1$$

极差由于不同观测数据的单位不同，因而不能用来比较两个不同样本。它只能粗略反映数据离散的程度，而不能反映数据的分布情况。极差对极端值极其敏感，因为只用两个极端的数据来衡量整组数据是不太科学的。为了能够比较不同样本的离散程度，需要进一步引入能够用于比较不同样本，衡量数据离散程度的度量。

（2）方差（Variance， VAR）就是样本中每个观测值 x_i 与样本算术平均值 $\bar{x}$ 之偏差的平方和，再除以样本容量减 1。其数学表达形式为

$$S^2=\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})^2$$

样本方差是描述数据取值分散程度的一个度量，它基于数据相对于平均值的偏离程度的平方和。其中样本方差计算公式中的分母 $n-1$ 称为 “样本自由度”，其取值不是样本容量 n 是为了使样本方差 S^2 是总体方差 σ^2 的无偏估计。当数据是 “总体” 数据而不是样本数据时，该自由度为 n；当数据为“样本”数据时，该自由度应该是 $n-1$。

在统计学中自由度（Degree of Freedom）是指计算某统计量时，取值不受限制的变量个数。通常自由度 $df=n-k$，其中 n 为样本容量，k 为限制条件的数目。对于单个变量（$k=1$）的样本数据，样本自由度 $df=n-k=n-1$，故而方差公式中为 $n-1$，而计算总体方差时则是除以 n。比如，某样本数据为 1,2,3,4，样本容量为 4，样本均值为 2.5，则样本受到总体均值 $\bar{x}=2.5$ 的约束。此时当样本中个体 1,2,3 一旦确定后，第 4 个值只能是 4，否则就不符合总体均值取值的约束，这就是为什么样本自由度取值 $df=n-k=4-1=3$ 的原因。

思考：如何在计算机编程时仅遍历样本数据一次就计算出样本方差？

$$S^2=\frac{n}{n-1}\left(\frac{1}{n}sum\left(x_i^2\right)-\left(\frac{1}{n}sum\left(x_i\right)\right)^2\right)$$

即

$$S^2=\frac{n}{n-1}\left(\frac{1}{n}\sum_{i=1}^{n}x_i^2-\left(\frac{1}{n}\sum_{i=1}^{n}x_i\right)^2\right)$$

（3）标准差 (Standard Deviation，SD) 为样本方差 S^2 的开方，又称为均方差或样本标准差。其数学表达形式如下：

$S=\sqrt{S^2}$ 对于总体则用 $\sigma=\sqrt{\sigma^2}$ 表示。

由于方差包含样本观测数值的平方，它的取值与数据本身因平方后相差太大，导致我们难以直观衡量数值的变异，因此需要引入样本标准差的概念。样本标准差能客观准确地反映样本数据的离散程度，在样本标准差几何上就是 n 维空间中的若干数据点去中心化后到一条直线的距离函数，称为二阶中心矩阵。

对于服从正态分布的数据，标准差在钟形曲线中对数据的分布具有非常重要的指示意义。如图 18-5 所示中间深色的两部分，它表示距离均值 μ 正负 1 个单位 σ 内的概率，即曲线下方的面积占总体概率 1 的 68.2%； 而距离 μ 正负 2 个单位 σ 内的概率为 95.4%，而 $\pm3\sigma$ 则为 99.7%，这 3 个特殊的概率值合称 68-95-99.7 准则或 3σ 准则，西

格玛准则在质量控制方面具有重要而广泛的意义。

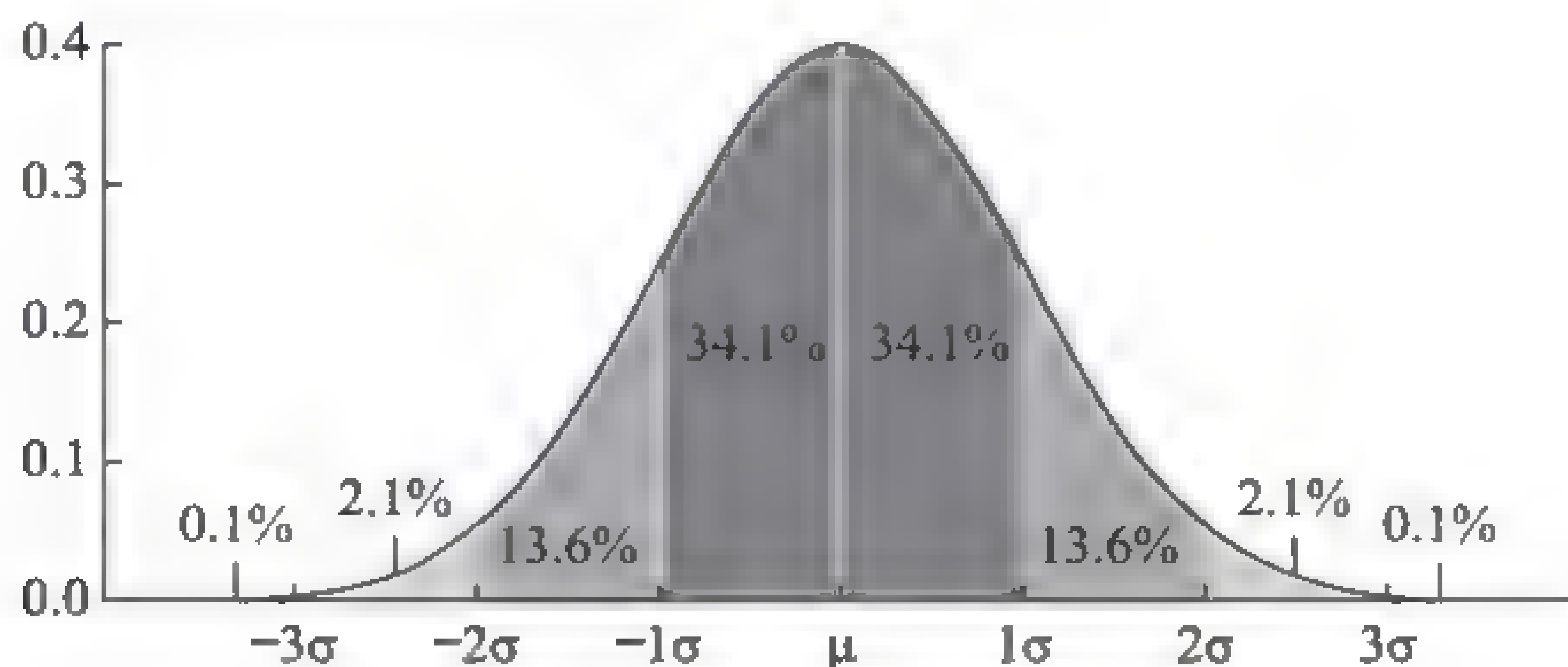


图 18-5　正态分布的 3σ 准则

标准误差（Standard Error，SE）也称标准误，与标准差概念不同。由于总体可以抽样出多个样本，而基于样本计算的统计量都是对总体数据的估计和反映，因此单个样本对总体数据的估计就存在误差，这种误差被称为抽样误差。而样本统计量本身的标准差称为标准误差，是描述样本统计量抽样分布的离散程度和误差大小的尺度。

比如，样本均值 $\bar{x}$ 这一统计量，尽管它是总体均值 μ 的无偏估计（Unbiased Estimates），但对一个总体的多次抽样（每次样本大小都是 n）时，则每个样本都有自己的均值 $\bar{x}_i$(第 i 次抽样计算出的均值)，这些平均值 $\bar{x}_i$ 的标准差就是样本均值 $\bar{x}$ 的标准误差（Standard Error of the Mean, SEM）。样本容量 n 越大，样本统计量的标准误差 SE 就越小，表示抽取样本越多，统计量能够反映总体的特征越好。样本均值 $\bar{x}$ 的标准误差为

$$\text{SEM}=\sigma/\sqrt{n}$$

它等于标准差除以样本容量 n 的开平方。

注意标准差是单次抽样计算所得，但用单次抽样得到的样本标准差 S，可以估计多次抽样才能估计总体的标准误差 σ。也就是说，样本标准差反映的是数据点的波动情况，而标准误差 SE 反映的是统计量均值的波动情况。只要样本总量趋向于无穷大，统计学的中心极限定理（Central Limit Theorem）可以保证其样本标准误差的样本分布渐进趋向于正态分布。

对于一个正态分布的总体来说，用样本均值估计总体均值需要考虑样本均值的方差或标准差，其数学表达形式为

- 样本均值的方差 $D(\bar{X})=\dfrac{D(X)}{n}$，即样本方差 S^2 除以 n。
- 样本均值的标准差 $SD(\bar{X})=\sqrt{D(\bar{X})}=\dfrac{SD(X)}{n}$，即样本标准差 S 除以 $\sqrt{n}$

（4）变异系数（Coefficient of Variation）定义为样本方差除以样本均值，它是一个消除了量纲的可比较不同样本的系数。其数学表达形式为

$$CV=\frac{S}{\bar{x}}\times 100\%$$

总体记为 $CV=\dfrac{\sigma}{\mu}\times 100\%$

标准差 S 固然能衡量数据的分散程度，但由于不同数据的数值大小和单位不同，它们之间就没有可比性。因此我们需要将标准差除以样本均值 $\bar{x}$ 来消除量纲的影响，这就

是需要变异系数 CV 的根本原因。

比如，我们要对比中国股市和美国股市的稳定性，美国股市上万点，而中国股市才 3000 点，美国股市波动 100 点跟中国股市波动 100 点是完全不同的，对于这种情况，我们就需要使用指数样本的变异系数 CV 进行比较，才能客观评价两个市场的波动性或稳定性的异同。

18.1.3　分布特征度量

衡量样本数据分布特征主要包括数据分布的对称性、分布曲线在中间聚集的陡缓程度两个方面，分别用偏度系数和峰度系数衡量。

（1）偏度（Skewness）用于衡量数据分布的对称性，其数学计算公式为

$$\beta_s = \frac{n^2}{(n-1)(n-2)} \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^3}{\sigma^3} = \frac{n^2}{(n-1)(n-2)} \frac{u^3}{\sigma^3}$$

其中 $u_3 = \frac{1}{n}\sum_{i=1}(x_i - \overline{x})^3$ 为 3 阶中心矩，即所有观测值与均值偏差的三次方总和的均值，σ^3 则为标准差的立方，对于样本数据则使用样本标准差 S^3 进行计算。为了确保样本的偏度系数是总体偏度的无偏估计，其前置系数为 $n^2/(n-1)(n-2)$。

若以正态分布为基准，如果样本数据计算出来的偏度不为零，则说明数据分布是有偏向性的。由于中位数永远是将样本容量分成相等的两半，如果平均值与中位数相等表示没有偏向性，而均值偏离中位数则会导致数据分布的不对称性（见图 18-6）。

- 偏度为负表示左偏（左拖尾），即左侧有更多的分散数据，此时在分布图上表现为算术平均数<中位数<众数的特征，均值在中位数的左侧（见图18-6左）
- 偏度为零表示对称分布，此时在分布图上有众数＝中位数＝平均数，3个统计量对应的观测值x重合（见图18-6中）。
- 偏度为正表示右偏（右拖尾），即右侧具有更多分散的数据，此时分布图上呈现众数 <中位数<算术平均数的特征，均值在中位数的右侧（见图18-6右）

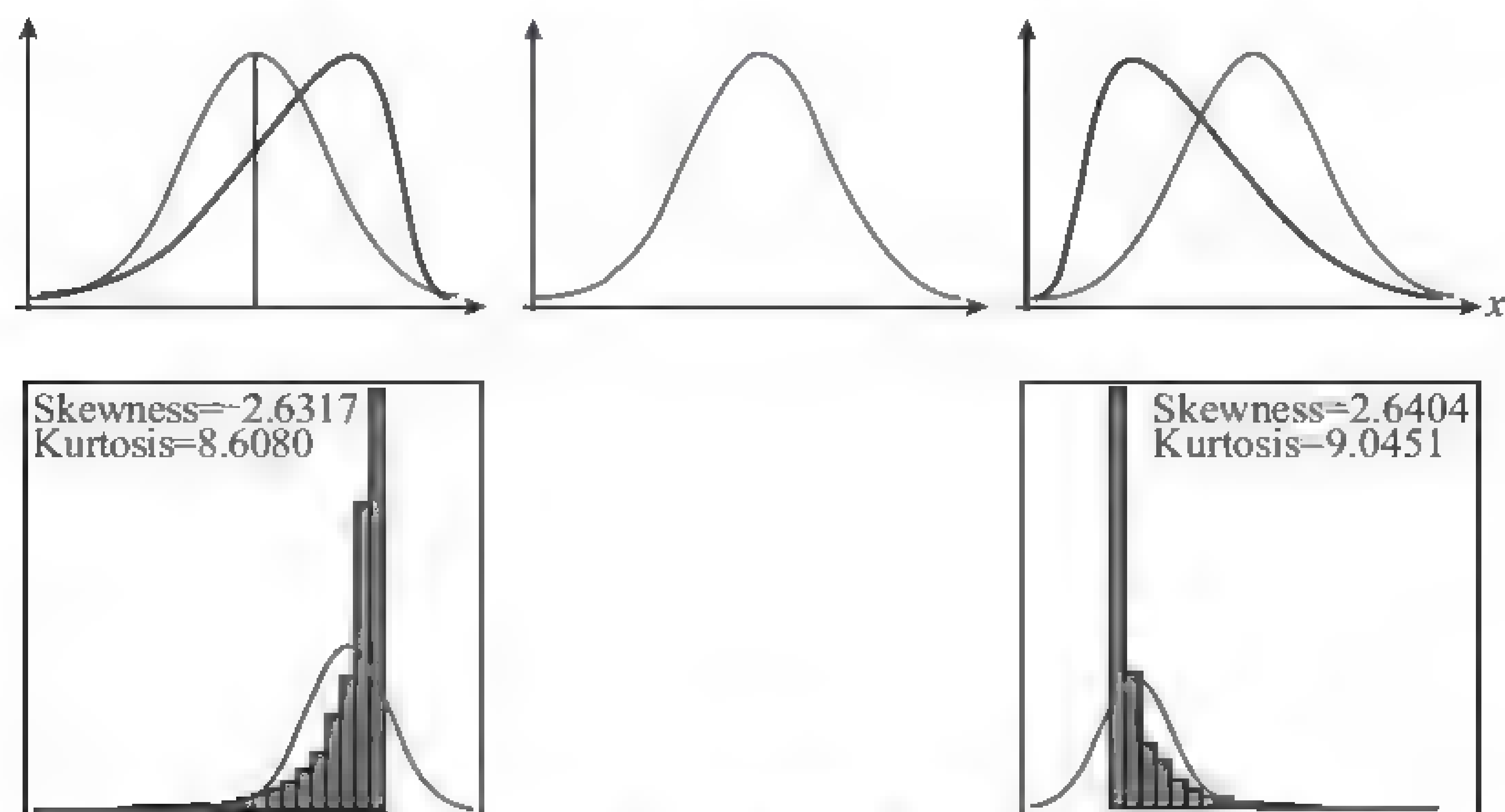


图 18-6　左偏与右偏分布

思考：如何在编程中仅用一次遍历样本数据就计算出偏度系数和峰度系数？

（2）峰度（Kurtosis）用于衡量数据分布形态的陡缓程度，也就是数据在中间聚集的程度。峰度描述所有数据取值分布形态的陡缓程度。峰度系数与变量四阶中心距和方差的平方有关，但需要乘以样本容量 n 有关的系数并减去与样本容量 n 有关的偏移量。其数学计算公式为

$$\beta_k = \frac{n^2(n+1)}{(n-1)(n-2)(n-3)}\frac{\frac{1}{n}\sum_{i=1}^{n}(x_i-\overline{x})^4}{\sigma^4} - \frac{3(n-1)^2}{(n-2)(n-3)}$$

$$= \frac{n^2(n+1)}{(n-1)(n-2)(n-3)}\frac{u^4}{\sigma^4} - \frac{3(n-1)^2}{(n-2)(n-3)}$$

其中 $u^4 = \frac{1}{n}\sum_{i=1}^{n}(x_i-\overline{x})^4$ 为是四阶中心矩。为确保样本峰度是总体峰度的无偏估计，其前置系数为 $\frac{n^2(n+1)}{(n-1)(n-2)(n-3)}$，并且需要减去一个与样本容量 n 有关的偏移量 $\frac{3(n-1)^2}{(n-2)(n-3)}$。

峰度系数取值大于 1 小于 n，由于标准正态分布的峰度系数为 3，均匀分布的峰度系数为 1.8，因此实践中通常在计算峰度系数时采取减 3 处理，使统计分布的峰度正负刚好能反映它相对于标准正态分布的差异，其绝对值越大，表示与标准正态分布的差异程度越大（见图 18-7）。

- 峰度为负，表示分布曲线比正态分布曲线矮，两侧分散的数据较多，呈现为平峰长尾的分布特征（见图18-7中最低的曲线）。
- 峰度为 0 时，数据的分布为正态分布，即中间曲线的形状。
- 峰度为正，表示分布曲线比正态分布曲线较高，两侧分散的数据较少，呈现为尖峰短尾的分布特征（见图18-7中最高的曲线）。

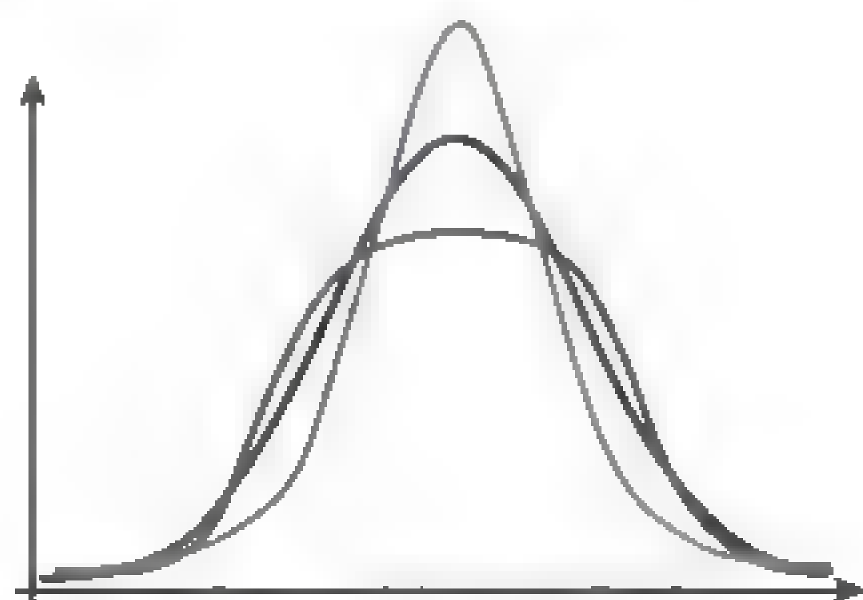


图 18-7　平峰长尾与尖峰短尾

数据分析中可用峰度系数这一统计量的标准误差来检验数据分布的正态性，如果峰度系数与其标准误差比值的绝对值较大，如该比值绝对值大于2，我们就可拒绝正态性。当然数据分布的正态性检验最常用的还是 Q-Q 图方法。

思考：总体正态分布的峰度系数为什么是 3？

$\beta_k = \frac{E\left[(X-\mu)^4\right]}{\sigma^4} = E\left[\left(\frac{X-\mu}{\sigma^4}\right)^4\right]$ 若 $X\sim N(u,\sigma^2)$，则 $z = X\sim N(u,\sigma^2)\sim N(0,1)$ 有

$$E(z^4) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} z^4 e^{-\frac{z^2}{2}}\mathrm{d}z = 3$$

偏度系数和峰度系数使用三阶和四阶中心矩进行计算是因为奥卡姆剃刀"如非必要，勿增实体"原则。在计算上更高阶中心矩固然可以采用，但它可能会导致复杂性太大而效果并不会比它们更好。偶数阶的中心矩反映分布的峰度，而二阶中心矩就是方差，那为什么不采用方差作为峰度的指标呢？那是因为有时由于方差相同的分布可具有不同峰度，所以需要引入更高阶的偶数中心矩，而下一偶数中心矩刚好为四阶中心矩；为了消除变量值和量纲的影响，它还需除以方差的平方。

18.1.4　置信区间、置信水平与 p-值

除了前面列出的这些样本统计量，还有很多其他统计量，如总和（SUM）、未校正的平方和（USS）、校正平方和（偏差平方和 CSS）、离差平方和（每个观测值与平均值之偏差的平方和 SSD）等。本节并不打算陷入这些计算细节之中，而是介绍 3 个核心统计学概念。

（1）置信区间（Confidence Interval，CI）：在正态分布曲线下方，围绕均值两边占比 95% 的区域，观测值 x 会落在介于 $\mu-1.96\sigma$ 和 $\mu+1.96\sigma$ 区间。换句话说，对于 x 服从均值为 μ，标准差为 σ 的正态分布，我们有 95% 的信心认为其观测值会落在 $\mu-1.96\sigma$ 到 $\mu+1.96\sigma$ 这一区间内。

由于统计量是基于样本计算出来的，因而基于每个样本计算出的统计量本身就是一个随机变量。而我们需要利用该统计量（是一个随机变量）推断出总体参数（一个实实在在的数值而非随机量），因此我们需要构造出一个包含"总体参数"真实值的区间，称为置信区间（置信区间本身也是随机变量），置信区间是"总体参数"的置信区间。

95% 的置信区间意味我们在同样的条件下重复抽样 100 次试验，基于 100 个样本计算出的置信区间里有 95 个是包括总体参数的真实值的，其余 5 个区间则不包含总体参数的真实值。要正确理解置信区间，**一定要记住样本统计量是随机变量，而总体参数是一个固定值**。图 18-8 中横坐标是试验次数 t，纵坐标是观测值 x，如 90% 的置信区间该如何理解？假如重复 10 次同样的试验，用 10 个样本计算得到的区间中有 9 个会包含观测的总体均值 μ，而其中某一个样本如图 18-8 中第 8 个样本则不包含总体均值 μ（不与总体均值 μ 相交）。

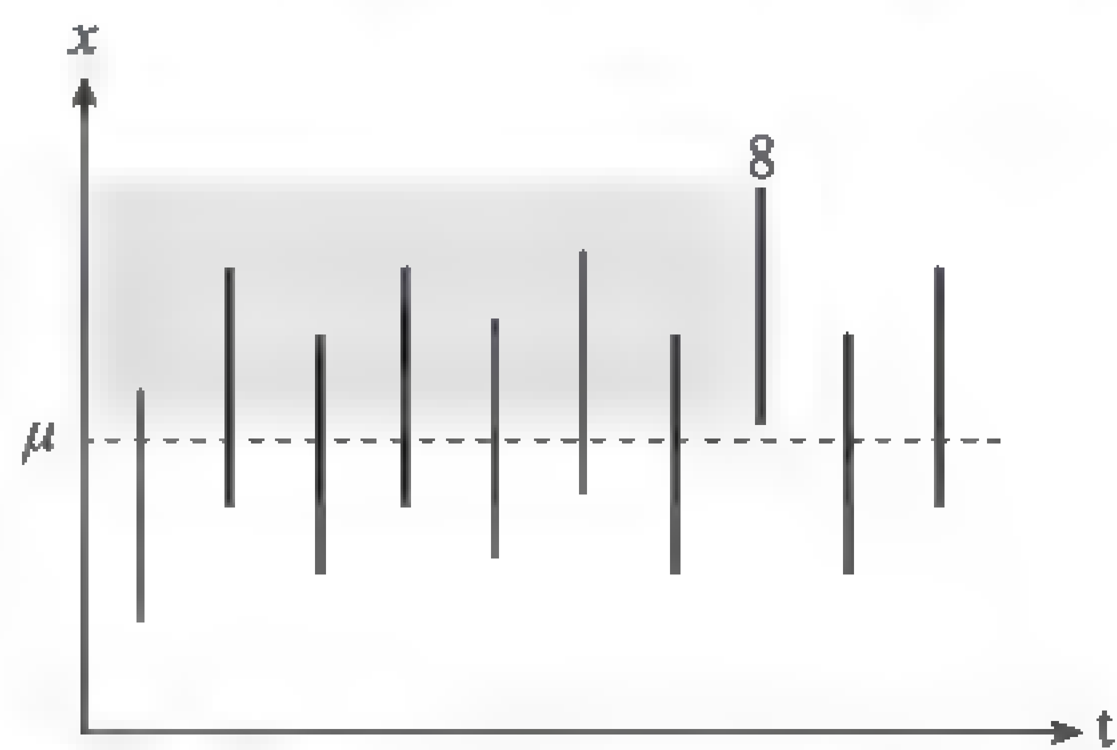


图 18-8　置信区间与总体均值 μ 的关系

（2）置信水平（Confidence Level）刻画总体均值 μ 落在置信区间 (a,b) 内的可能性大小，称为置信水平。而总体平均值 μ 落在置信区间外的可能性称为显著性水平，用 α 表示，则置信水平为 $1-\alpha$，置信水平和显著性水平之和恒为 1。

在置信水平相同的情况下，样本容量n越大，置信区间越窄；样本容量相同的情况下，置信水平越高，置信区间越窄。置信水平95%意味着多次抽样中有95%的机会置信区间包含未知的总体参数值，而另外的5%则不包含总体参数的真实值。至于基于单次抽样计算得到的置信区间是包含总体参数真实值的区间，还是属于个别不包含参数值的区间则不得而知。

（3）P-值（P-Value）是统计学家费歇在假设检验理论中引入，反映原假设H_0发生概率的概率值。它就是在一次假设检验中，利用观测值能够做出拒绝原假设H_0的最小显著性水平。

由于显著性水平α是人为给定的阈值，它无法反映出观察到的数据与原假设H_0之间不一致的程度。为了衡量原假设H_0为真，所得到的样本观察结果或更极端的结果出现的概率，也就是说检验统计量大于或等于实际观测值的概率，需要引入P-值来表示H_0能被拒绝的最小显著性水平，因此P-值也被称为观察到的显著性水平。

P-值跟显著性水平α非常类似但千万不能混淆。一般而言，如果P-值大于0.05，则表示倾向于接受假定的总体参数取值；如果P-值小于0.05，则表示必须拒绝假定的总体参数取值；如果P-值小于0.01，则表示较强的判定结果，否则表示较弱的判定结果。

18.2 统计学上的变量类型

在从数据到智能的演进过程必须经历4个阶段：数据→信息→知识→智能，其中数据扮演着最基础的角色。数据本身来自实际的观测，它是最为原始的实体，数据本身除了表示一种存在性外并没有任何含义，通常表现为一些符号，数据就是一些字符或者数字组成的实体。数据库就是包含一系列由这些数字或符号组成的所谓数据库表或者树状层次条目。

在统计学上根据观测数据所包含的信息，可以将由字符或数值组成的数据分成分类数据（Categorical）和定量数据（Quantitative），两种数据代表了数据量化过程中的定性和定量两个阶段。在计算机上数据是特定变量的取值，其中分类变量可以由字符类型（Character）或者数值（Numeric）类型数据构成，而定量变量只能由数值类型数据构成。

有很多计算机专业人士刚开始学习SAS语言编程时，总是会惊异于SAS的DATA步中为什么只有这两种最基本的数据类型，而不像其他计算机编程语言那样包括布尔型、字节型、短整型，长整形、单精度浮点数、双精度浮点数等各种基本数据类型。其根本的原因是SAS语言从定义之初就是为数据分析科学而设计的，它关注数据所能表达的信息量，甚于数据的存储和运行结构。实际上，大部分通用编程语言的数据类型系统都是面向计算机存储和表达，而不是数据中所能够表达的语义。

对于分类变量，它可以进一步根据测量尺度（Scale of measurement）或测量水平而分为定类变量（Nominal）和定序变量（Ordinal）；而定量变量根据观测值出现的特征可分成离散变量（Discrete）和连续变量（Continual），其中对于连续变量，根据测量尺度可以进一步分成定距变量（Interval）和定比变量（Ratio）。统计学上我们会探讨各种

随机变量以及它们的概率分布，大部分是对定量变量而言的。统计学上根据测量尺度建立的数据类型系统如图 18-9 所示。

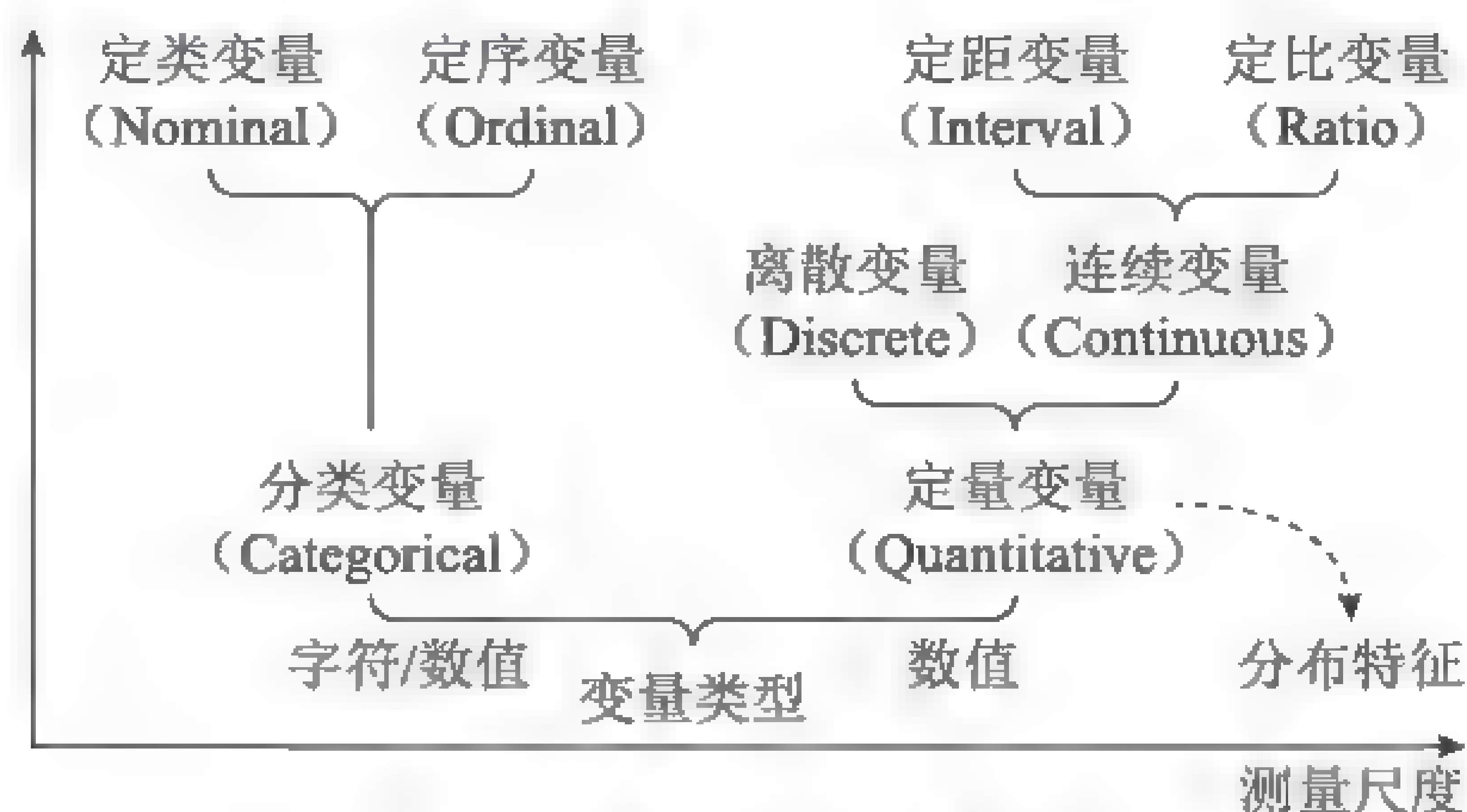


图 18-9　统计学上的变量类型

既然数据本身除了存在别无意义，我们从这些字符或者数值的“存在”中辨别它们的统计学上的角色就很重要，因为不同的测量尺度下，它们所能表达的彼此之间的差异，以及差异的多少是不同的。同样是 1 和 3，如果表示的是运动员球衣上的编号，它只是辨别不同运动员的身份的定类变量；但如果 1，2，3 分别代表冠军、亚军和季军排名，则 1 和 3 是定序变量，其中蕴含着第 1 名比第 3 名更优秀的信息，然而它并不能揭示他们之间“优秀”的差别到底有多大。

如果 1，2，3 是十二小时制（1:00:00-12:59:59）计量的时钟时间，即 1 点钟、2 点钟和 3 点钟，则它能为我们提供更多的信息：1 点钟和 2 点钟之间相差 1 个小时，2 点钟和 3 点钟之间也是相差 1 个小时，两者之间的差别是相等的，这种差别跟 12 点和 1 点之间的差别也是一样的，此时十二小时制计量的时钟时间就是一个定距变量，我们不能认为 12 点钟和 3 点钟的差别跟 9 点钟和 12 点钟的差别有什么不同，此时数据中绝不包含这么一个信息：12 点钟是 3 点钟的 4 倍，而 9 点钟是 12 点钟的 3/4。按摄氏度计量的温度也是定距变量的一个例子，如 1℃和 3℃，它们相差 2℃。在人类发现热力学绝对零度（-273.15℃）的存在之前，我们只能用摄氏度进行温度计量。

如果 1，2，3 是二十四小时制（0:00:00-23:59:59）计量的时钟时间，即 1 点钟、2 点钟和 3 点钟。看起来似乎它跟十二小时制之间没有什么区别。然而，该测量尺度下蕴含一天的起点是零点的信息，因此 1 点钟和 12 点钟的差别是 11 个小时，而不像十二小时制下相差 1 个小时。用绝对温度（以 K 为单位）计量的温度 1K 和 3K 由于存在绝对零度，它们隐含了更多信息：理想气体在绝对温度 3K 时的体积约为是 1K 时的 3 倍。因此如果数据是用二十四小时制或绝对温度计量的，则变量变成了定比变量。因此纯粹用数字表达信息是不充分的，**在数据分析时最重要的是要知道这些数据所蕴含的信息量有多大，而变量所包含的信息量跟测量尺度紧密相关。**

表 18-1 列出了统计学上根据测量尺度划分的变量类型，越高级的变量类型刻画对象的能力越强，它具有较低级变量的特征，也可以退化为较低级别的变量类型进行数据分析处理，但低级变量类型不能被误用为高级变量类型进行分析，因为那是

毫无意义的。

表 18-1　统计学上的变量类型

类　型	描　述	实　例
定类变量	用字符或数值表示个体在属性上的特征或类别上的不同，是一种分类标志，但没有顺序和大小之分。只是按事物的某种属性对它进行分类或分组，能衡量事物之间类别上有差异，却不知道它们的差异有多大。因此定类变量只能作“=”与“≠”比较运算。如果定类变量只有两种可能的差异，则称为二项变量（Binominal）	如二项变量性别：男 / 女其他如国籍、信仰、人种等
定序变量	用字符或数字表示个体在某个有序状态中的位置，它不但能描述事物之间存在差别，而且能够描述等级或顺序上的差异，所以又称顺序尺度。定序变量除了拥有定类变量的性质（“=”与“≠”比较运算）外，它由于有方向性，所以可作“<”和“>”比较运算。然而，定序变量也仅此而已，它不能衡量差别的程度，它依然是一个没有量化信息的分类变量	如学历：高中 / 大学 / 研究生；其他如收入等级、职称级别、冠亚季军、左中右等
定距变量	它是用数值表示的，能够衡量彼此之间间距的变量，也称间隔尺度；但由于该尺度下没有一个用作比较标准的绝对零点，因此它只能衡量事物在类别或次序上的距离，通常带有自然或物理上的计量单位，如摄氏度衡量的温度就是一个定距变量 定距变量除了有定类变量和定序变量的性质外，它可以做加减运算，但不能乘除。比如 12 小时制（1~12 点）下的时钟时间：9 点不代表 3 点的 3 倍，而是代表它们之间间隔 6 小时	如按照摄氏度计量的温度，20 ℃ 和 30 ℃ 相差 10℃；十二小时制计量的时间 1 点和 12 点之间相差 1 个小时
定比变量	该测量尺度下的变量除了具有定距变量所具有的性质外，它还具有表示没有或理论极限的基准零点。因此基于基准零点，定比变量不但可以作加减运算，还可以作乘除运算。比如身高、尺寸、体积，收入金额等。它的取值通常大于零，定比尺度变量取对数就变成了定距尺度变量	如身高、尺寸、体积，用 K 为单位的绝对温度、反应时间、收入、考试成绩、二十四小时制时间等

辨别统计学上的数据类型非常重要，因为它不但决定了可以用什么统计量，绘制什么类型的图形图表，还决定了用哪一种统计分析方法进行数据处理。频数、百分比以及众数是适用于任何数据类型进行分析的统计量，而中位数、四分位数 / 百分位数则对定序类型数据及以上的定量数据类型有效。均值和方差适用于所有的定量数据类型，而几何平均数、调和平均数只能用于定比数据类型。也就是说，**不同测量尺度下的数据类型所蕴含的信息量（或者信息熵）是不同的，分析数据的第一步就是要搞清楚数据的测量尺度，然后根据数据类型选择适合的统计分析方法**。表 18-2 列出了不同变量类型和适用的基本统计量之间的关系。

表 18-2　不同变量类型与适用的基本统计量

类　型	适用的基本统计量			
定类变量	频数、百分比、众数			
定序变量	频数、百分比、众数	中位数，分位数		
定距变量	频数、百分比、众数	中位数，分位数	均值，方差	
定比变量	频数、百分比、众数	中位数，分位数	均值，方差	几何，调和平均数

18.3 基本数据处理

SAS 对各种数据分析功能进行了非常好的封装，因此大部分功能都是只需要提供输入数据集，设置正确的参数调用对应的 PROC 步进行处理即可。在 SAS 里进行数据分析首先是需要了解数据的基本结构和类型，SAS 使用逻辑库和数据集来组织数据，因此我们首先可调用 PROC DATASETS 来了解某个逻辑库的信息，程序 18-1 列出系统逻辑库 SASHELP 有关的各种基础信息，比如 SASHELP 逻辑库被映射到 23 个不同的目录路径上，共包含 408 个数据成员。

程序18-1 列出SASHELP 逻辑库信息

```
proc datasets library=sashelp ;
quit;
```

其输出如图 18-10 所示。

Level 23	
引擎	V9
物理名	C:\Program Files\SASHome\SASFoundation\9.4\stat\sashelp
文件名	C:\Program Files\SASHome\SASFoundation\9.4\stat\sashelp
所有者名	BUILTIN\Administrators
文件大小	4KB
文件大小（字节）	4096

#	名称	成员类型	水平	文件大小	上次修改时间
1	AACOMP	DATA	4	384KB	2016-11-10 14:36:30
	AACOMP	INDEX		161KB	2016-11-10 14:36:30
2	AARFM	DATA	3	256KB	2016-11-10 14:39:26
	AARFM	INDEX		25KB	2016-11-10 14:39:26
3	AC	CATALOG	3	17KB	2016-11-10 14:44:59
406	ZIPMIL	DATA	16	648KB	2016-11-10 14:47:55
	ZIPMIL	INDEX		45KB	2016-11-10 14:47:55
407	ZTC	DATA	3	576KB	2016-11-10 14:39:31
408	_CMPIDX_	DATA	8	192KB	2016-11-10 14:39:54

图 18-10 查看逻辑库内容

如果想在查询数据集信息时顺便把某个数据集（数据表）的详细信息列出来，可使用该过程步的 contents 语句指定数据集的名字（见程序 18-2）。

程序18-2 列出SASHELP 逻辑库信息和数据表CLASS 的元数据信息

```
proc datasets library=sashelp ;
  contents data=SASHELP.CLASS order=collate;
quit;
```

如果我们只想知道某个数据集 SASHELP.CLASS 的属性信息，即数据集的元数据，使用 PROC CONTENTS 即可，程序 18-3 可输出与 PROC DATASETS 指定 CONTENTS 语句的输出相同结果。

程序18-3 列出SASHELP.CLASS 数据表的元数据信息

```
proc contents data=sashelp.class;
run;
```

如果我们需要查看数据集中的数据本身，则可以通过 PROC PRINT 过程步来控制，程序 18-4 打印结果时只选择显示 2 个变量（性别和身高）的信息，而且打印的时候只选前 10 行记录。如果变量有标签信息，则显示标签（列的说明文字）。

程序18-4 打印数据集部分变量的信息，仅限前10 行

```
proc print data=SASHELP.CLASS(obs=10) label;
  var Sex Height;
run;
```

如果觉得 PROC PRINT 功能不够强大，可以使用 PROC REPORT 过程步来输出更加复杂的报告（见程序 18-5），比如定制标签和格式，并在底部显示汇总行（见图 18-11）。

程序18-5 PROC REPORT 生成定制格式报告

```
proc report data=sashelp.class;
  where Age>12;
  format name  $10. sex $2. weight  7.2  height 7.2;
  label name="大名";
  rbreak after / summarize style=[font_weight=bold];
run;
```

大名	性别	年龄	身高（英寸）	体重（磅）
阿尔弗雷德	男	14	69.00	112.50
凯露	女	14	62.80	102.50
亨利	男	14	63.50	102.50
雅妮特	女	15	62.50	112.50
茱迪	女	14	64.30	90.00
玛丽	女	15	66.50	112.00
菲利普	男	16	72.00	150.00
罗纳德	男	15	67.00	133.00
威廉	男	15	66.50	112.00
		132	**594.10**	**1027.00**

图 18-11 PROC REPORT 输出报告

如果对于某个逻辑库具有写权限，比如临时逻辑库 WORK 中的数据，我们可以随时删除不需要的数据集（见程序 18-6）。

程序18-6 删除数据集

```
proc delete data=work.MYCLASS;
run;
```

18.3.1 排序与排名

数据集可根据一个或多个变量进行排序，输出唯一的排序结果。默认对每个变量递增排序，如果要进行递减排序，则需要在对应的变量名前面加上关键词 DESCENDING。排序时也可指定多个变量依次进行排序（见程序 18-7）。

程序18-7　数据排序：可以按多列排序，可在变量前加上关键词DESCENDING

```
proc sort data=sashelp.class out=sort_class ;
  by  height  descending weight;
run;
proc print;run;
```

排名与排序是不同的概念，它是计算某个变量的观测在整个数据集中的位置排名，每个数值变量都可以计算对应的排名。程序 18-8 将 SASHELP.CLASS 按照身高排名，形成排名变量 rank Height 并将结果输出到 Work.rank class 中。排名 RANK 与排序 SORT 不同，数值相同的变量会共享排名，它等于校正前的排名之和除以相同排名的观测个数。比如身高为 62.5 英寸的雅妮特与亨利排名后都占据第 8 位和第 9 位，因此他们共享排名 (8+9)/2 = 8.5，同理身高为 66.5 英寸的玛丽和威廉共享排名 15.5。比如：

程序18-8　数据排名：按照身高进行排名求秩

```
proc rank data=sashelp.class out=rank_class;
  var height;
  ranks rank_height;
run;
```

程序 18-9 将输出数据根据排名变量 rank_Height 进行排序后打印，它可检查排序与排名之间的不同。

程序18-9　按排名变量对观测进行排序检查

```
proc sort data=rank_class out=sort_rank_class;
  by rank_height;
run;
proc print;run;
```

也可以同时为多个变量生成对应的排名变量，比如同时输出 Height 和 Weight 的排名值（见程序 18-10），其输出结果如图 18-12 所示。

程序18-10　对多个变量进行排名生成对应的秩

```
proc rank data=sashelp.class out=rank_class;
  var height weight;
  ranks rank_height rank_weight;
run;
```

Obs	Name	Sex	Age	Height	Weight	rank Height	rank Weight
1	阿尔弗雷德	男	14	69.0	112.5	18.0	15.5
2	爱丽丝	女	13	56.5	84.0	3.0	4.5
3	芭芭拉	女	13	65.3	98.0	14.0	9.0
17	罗纳德	男	15	67.0	133.0	17.0	[illegible]
18	托马斯	男	11	57.5	85.0	5.0	7.0
19	威廉	男	15	66.5	112.0	15.5	13.5

图 18-12　生成变量的秩

18.3.2　数据转置

将数据的行和列进行交换称为数据转置，默认不用 VAR 语句指定变量时只对数值

变量进行转置，否则需要人为用 VAR 语句指定需要转置的列。数据转置时也可以指定输出列名的前缀。考察如下程序 18-11，它对数据集的行列进行了旋转。结果如图 18-13 所示。

程序18-11　默认数据转置，仅对数值变量起作用

```
proc transpose data=sashelp.class out=tran class;
run;

proc transpose data=sashelp.class out=tran_class prefix=var;
  var name age sex height weight ; /*显式指定变量*/
run;
proc print;run;
```

图 18-13　数据转置

18.3.3　堆叠与拆分

有时需要将数据集中两个或多个列堆叠起来，形成更多的数据行。比如我们可以把具有 19 行观测的 SASHELP.CLASS 中的 5 个列堆叠起来（见程序 18-12），形成 95 (=19×5) 行的数据集。结果如图 18-14 所示。

程序18-12　数据堆叠

```
data class;
  set sashelp.class;
  id=_n_;/*生成行标识*/
run;

proc transpose data=class out=stack_class(drop=_name_ _label_
  rename=(col1=all))  ;
  var name age sex height weight; /*堆叠 5 列为 1 列，列名为 all*/
  by id;
run;
proc print; run;
```

图 18-14　数据的堆叠

而有时候需要根据某个变量的取值，将数据拆分成多个数据行或列。比如按照性别不同将变量身高的数据变成两行（见程序 18-13）。注意其中由于女性的数据比男性的数据少，因此第 2 行最后一列为缺失值（见图 18-15）。

程序18-13　数据拆分

```
proc sort data=sashelp.class out=sort_class;
  by sex ;
run;

proc transpose data=sort_class out=split_class(drop=_name_) prefix=VAR;
  var height;
  by sex;
run;
proc print;run;
```

Obs	Sex	_LABEL_	VAR1	VAR2	VAR3	VAR4	VAR5	VAR6	VAR7	VAR8	VAR9	VAR10
1	男	身高（英寸）	69.0	63.5	57.3	62.5	59.0	72.0	64.8	67.0	57.5	66.5
2	女	身高（英寸）	56.5	65.3	62.8	59.8	62.5	51.3	64.3	56.3	66.5	

图 18-15　数据拆分

如果仅仅是为了根据性别把数据拆分表成两个数据表 ClassF 和 ClassM，我们还可通过如下程序进行拆分（见程序 18-14），结果如图 18-16 所示。

程序18-14　将数据按照某个变量拆分成两个表

```
data ClassF(where=(sex='男')) ClassM(where=(sex='女'));
  set sashelp.class;
run;
proc print data=ClassF;run;
proc print data=ClassM;run;
```

Obs	Name	Sex	Age	Height	Weight
1	阿尔弗雷德	男	14	69.0	112.5
2	亨利	男	14	63.5	102.5
3	詹姆斯	男	12	57.3	83.0
4	杰弗瑞	男	13	62.5	84.0
5	约翰	男	12	59.0	99.5
6	菲利普	男	16	72.0	150.0
7	罗伯特	男	12	64.8	128.0
8	罗纳德	男	15	67.0	133.0
9	托马斯	男	11	57.5	85.0
10	威廉	男	15	66.5	112.0

Obs	Name	Sex	Age	Height	Weight
1	爱丽丝	女	13	56.5	84.0
2	芭芭拉	女	13	65.3	98.0
3	凯露	女	14	62.8	102.5
4	简	女	12	59.8	84.5
5	雅妮特	女	15	62.5	112.5
6	乔伊斯	女	11	51.3	50.5
7	朱迪	女	14	64.3	90.0
8	罗伊斯	女	12	56.3	77.0
9	玛丽	女	15	66.5	112.0

图 18-16　表格拆分

18.3.4　过滤数据

在 SAS 中过滤数据可以有多种方法，最简单的是利用 DATA 步的内置 WHERE 或 IF 语句，或者 WHERE 选项。比如按照值过滤出身高大于 60 英寸的记录，可用如下几种方法实现（见程序 18-15）。

程序18-15 WHERE语句和IF语句过滤数据

```
data Filter_Class;
  set sashelp.class;
  where Height>60;
run;

data Filter_Class;
  set sashelp.class;
  if Height>60;
run;
```

通过 WHERE 语句和 IF 取子集语句都可实现对数据的过滤，然而 WHERE 语句在数据进入 PDV 之前就进行过滤，是最为有效的；而用语句 IF 过滤，是对已经加载到 PDV 中的数据进行过滤，因此效率稍差。另外，WHERE 是编译型语句，仅适用于从 SAS 数据集读入的变量，而 IF 是可执行语句，对 SAS 数据集和用 INPUT 语句从外部文件读入的变量均有效。实际上，下例中 IF 取子集 Height>60 等价于 if Height<=60 then delete 语句（见程序 18-16）。

程序18-16 IF 语句的两种等价形式

```
data Filter_Class;
  set sashelp.class;
  if Height<=60 then delete; /*等价于 IF 取子集*/
run;

data Filter_Class;
  set sashelp.class;
  if Height>60 then output;
run;
```

更灵活的是在指定输入和输出数据时通过选项进行指定，程序 18-17 中第一段代码直接在输入时就进行过滤，而第二段在最后输出到数据集时进行过滤。

程序18-17 使用输入／输出数据集选项过滤数据

```
data Filter_Class;
  set sashelp.class(where=(Height>60));
run;

data Filter_Class(where=(Height>60));
  set sashelp.class;
run;
```

当然，我们也可以用 SQL 语句对数据集进行过滤（见程序 18-18），结果是等价的。

程序18-18 使用PROC SQL过滤数据

```
proc sql noprint;
  create table Filter_Class as
    select * from SASHELP.CLASS where(Height >60);
run;
quit;
```

在执行统计分析时，有时还需要根据比例或者百分位数进行过滤，比如过滤出身高排名在中间 50% 的记录。此时我们就需要先计算出 Q1、Q3 百分位数，然后根据百分位数 (Q1 ＜=Height ＜ Q3) 进行过滤（见程序 18-19）。

程序18-19　使用分位数过滤数据

```
proc univariate data SASHELP.CLASS noprint;
  var Height;
  output pctlpts=25 pctlpts=75 pctlpre=Height out=HeightPct;
run;
data _null_;
  set HeightPct; /*利用宏变量在 DATA/PROC 之间传递数值*/
  call symput('_Height25', Height25);
  call symput('_Height75', Height75);
run;

proc sql noprint;
  create table WORK.filter as
    select * from SASHELP.CLASS
      where( &_Height25<= Height AND Height <&_Height75)
        order by Height; /*利用传递过来的宏变量进行过滤*/
quit;
proc print;run;
```

18.3.5　随机抽样

由于总体的不均匀性和样本的随机性，从总体中抽取的个体通常不能精确反映总体。因此统计学中发展出各种抽样设计方法，以确保抽出来的样本具有很好的代表性。表现在数据分析中就是如何从一个大的数据集中科学地抽取一个较小的样本，主要有简单随机抽样、周期系统抽样、分层抽样和整群抽样等方法。

如要对数据集进行抽样，需要先将数据按变量排序后使用 PROC SURVEYSELECT 来抽样。SAS 提供各种各样的抽样方法。程序 18-20 使用简单随机抽样 (Simple Random Sampling，SRS) 方法从 SASHELP.CLASS 19 个变量中根据性别 Sex 变量等概率无替换地抽取 10 个样本，结果如图 18-17 所示。

程序18-20　数据简单随机抽样示例

```
proc sort data=SASHELP.CLASS out=Sort_Class;
  by Sex;
run;

proc surveyselect data=Sort_Class out=Sample_Class method=srs sampsize=10 ;
  strata Sex / alloc=prop;
run;
proc print; run;
```

Obs	Sex	Name	Age	Height	Weight	Total	AllocProportion	SampleSize	ActualProportion	SelectionProb	SamplingWeight
1	男	阿尔弗雷德	14	69.0	112.5	10	0.52632	5	0.5	0.50000	2.0
2	男	亨利	14	63.5	102.5	10	0.52632	5	0.5	0.50000	2.0
3	男	杰弗瑞	13	62.5	84.0	10	0.52632	5	0.5	0.50000	2.0
4	男	约翰	12	59.0	99.5	10	0.52632	5	0.5	0.50000	2.0
5	男	罗伯特	12	64.8	128.0	10	0.52632	5	0.5	0.50000	2.0
6	女	爱丽丝	13	56.5	84.0	9	0.47368	5	0.5	0.55556	1.8
7	女	芭芭拉	13	65.3	98.0	9	0.47368	5	0.5	0.55556	1.8
8	女	凯露	14	62.8	102.5	9	0.47368	5	0.5	0.55556	1.8
9	女	雅妮特	15	62.5	112.5	9	0.47368	5	0.5	0.55556	1.8
10	女	罗伊斯	12	56.3	77.0	9	0.47368	5	0.5	0.55556	1.8

图 18-17　数据抽样

如果要进行不受限制的简单随机抽样，应该将方法设置为 METHOD URS，它是有替换的等概率抽样。该过程步支持各种等概率随机抽样，只需要简单设置抽样方法即可实现简单随机抽样（无放回的）、不受限制的简单随机抽样（有放回）、系统随机抽样、顺序随机抽样以及伯努利抽样等。PROC SURVEYSELECT 也支持泊松抽样和各种 PPS 抽样（容量比例概率二阶段整群抽样）。

18.3.6 基本统计量

SAS 有很多过程步可以生成前面所讲的基本统计量，其中 PROC MEANS 就是最简单的方法。默认情况下它计算样本容量 N、均值、标准差、最小值和最大值。程序 18-21 至程序 18-24 分别计算 SASHELP.CLASS 数据集的基本统计量、集中趋势统计量、离散趋势统计量和分布特征统计量，输出分别如图 18-18 到图 18-21 所示。

程序18-21 计算指定变量的基本统计量：N 均值标准差最小值最大值

```
proc means data=sashelp.class;
  var Age Height Weight ;
run;
```

变量	标签	N	均值	标准差	最小值	最大值
Age	年龄	19	13.3157895	1.4926722	11.0000000	16.0000000
Height	身高（英寸）	19	62.3368421	5.1270752	51.3000000	72.0000000
Weight	体重（磅）	19	100.0263158	22.7739335	50.5000000	150.0000000

图 18-18 基本统计量（1）

程序18-22 计算数据集中趋势统计量：均值众数中位数

```
proc means data=sashelp.class mean mode median  ;
  var Age Height Weight ;
run;
```

变量	标签	均值	众数	中位数
Age	年龄	13.3157895	12.0000000	13.0000000
Height	身高（英寸）	62.3368421	62.5000000	62.8000000
Weight	体重（磅）	100.0263158	84.0000000	99.5000000

图 18-19 基本统计量（2）

程序18-23 计算数据离散趋势统计量：极差方差标准差变异系数最小值下四分位数上四分位数最大值

```
proc means data=sashelp.class range var std cv min q1 q3 max;
  var Age Height Weight ;
run;
```

变量	标签	极差	方差	标准差	变异系数	最小值	四分位数下限	四分位数上限	最大值
Age	年龄	5.0000000	2.2280702	1.4926722	11.2097909	11.0000000	12.0000000	15.0000000	16.0000000
Height	身高（英寸）	20.7000000	26.2869006	5.1270752	8.2247914	51.3000000	57.5000000	66.5000000	72.0000000
Weight	体重（磅）	99.5000000	518.6520468	22.7739335	22.7679419	50.5000000	84.0000000	112.5000000	150.0000000

图 18-20 基本统计量（3）

程序18-24 计算数据分布特征统计量：偏度和峰度系数

```
proc means data=sashelp.class SKEW KURT;
  var Age Height Weight ;
run;
```

变量	标签	偏度	峰度
Age	年龄	0.0636117	-1.1109255
Height	身高（英寸）	-0.2596696	-0.1389692
Weight	体重（磅）	0.1833510	0.6833648

图 18-21　基本统计量（4）

实际上，PROC MEANS 也可以计算均值 95% 的置信区间上下限 UCLM 和 LCLM，同时也可以计算 t 统计量以及对应的概率值（见程序 18-25），进行简单的假设检验。此时如果该变量表示两个样本的差异值，则计算出来的 t 统计量和 P- 值可直接进行 t 检验（见图 18-22 所示，详情参考方差分析一章）。

程序18-25　计算置信区间以及 t 统计量

```
proc means data=sashelp.class LCLM UCLM t Probt;
  var Age Height Weight ;
run;
```

变量	标签	均值95% 置信下限	均值95% 置信上限	t 值	Pr > |t|
Age	年龄	12.5963445	14.0352344	38.88	<.0001
Height	身高（英寸）	59.8656709	64.8080133	53.00	<.0001
Weight	体重（磅）	89.0496312	111.0030004	19.14	<.0001

图 18-22　均值的置信区间与 t 统计量

作为程序员与数据科学家，不仅要知道如何用 SAS 方便地算出这些统计量。有必要的时候还能自己写程序进行计算。下面笔者编写的 SAS 程序（见程序 18-26）仅用一次循环样本数据就可计算出偏度系数和峰度系数，结果与 SAS 计算出来的完全相同（如图 18-23）。

程序18-26　编程计算基本统计量：方差，偏度和峰度系数

```
data STAT_CLASS;
  set sashelp.class end=last;

  retain sum 0; retain sum2 0; retain sum3 0; retain sum4 0;
  sum=sum+weight;
  sum2=sum2+weight**2;
  sum3=sum3+weight**3;
  sum4=sum4+weight**4;

  if last then do;
    mean=sum/_N_;
    n=_N_;

    variance = sum2 / n - (mean **2 ); /*方差 Viriance, 如果是样本*/
    if n>1 then variance = variance * n / (n-1); /*如果是样本，则应该是 N-1 而非 N*/
    sd=sqrt( variance); /*标准差 Standard Deviation */

    skewness= sum3  - 3 * sum2 * mean  + 3 * sum * mean**2  - n * mean**3;

    skewness= skewness/n;
    skewness= skewness * n / (n-1) * n / (n-2) ;
    skewness= skewness / (variance * sd);

    kurtosis= sum4 - 4* sum3 * mean  + 6 * sum2 * mean**2 - 4 * sum *
      mean**3 +  n * mean**4 ;
    kurtosis= kurtosis / n;
```

```
      kurtosis=kurtosis * (n / (n 1))* (n / (n 2)) * ((n+1)/(n-3));

      kurtosis=kurtosis / (variance * variance);
      kurtosis=kurtosis - 3* ( n-1 )/(n-2) * (n-1)/(n-3);
      put skewness  9.7 " " kurtosis 9.7;
      output;
    end;
    keep n sum mean variance skewness kurtosis ;
    label n="观测的个数" sum="总和" Mean="均值"  Variance="方差" skewness="
      偏度" kurtosis="峰度" ;
run;
proc print data=STAT_CLASS label noobs;run;
```

总和	均值	观测的个数	方差	偏度	峰度
1900 5	100 026	19	518 652	0.18335	0 68336

图 18-23 编程计算偏度与峰度系数

18.4 基本图形图表

图形图表在直观显示数据分布特征，揭示数据潜藏的模式和识别异常值方面具有重要的作用，因此对于不同数据类型需要用不同的图形图表进行展现，称为数据可视化。进行数据可视化是需要根据前面讲的对不同测量尺度下的数据类型进行科学呈现，如果用不恰当的图形呈现数据可能引入没有意义的误导信息。

1. 柱状图/条形图

在 Base SAS 中可以有多种方法呈现图形，用得较多的是 SGPLOT 过程步，其中用不同的语句控制图表的类型，比如 vbar 表示绘制柱状图（垂直条形图），hbar 表示绘制条形图（水平条形图）。程序 18-27 绘制了简单的垂直条形图（见图 18-24）。

```
程序18-27  简单垂直条形图：分类变量为 Age，响应变量为 Height，默认统计量为 SUM
proc sgplot data=SASHELP.CLASS;
  vbar Age / response=Height;
run;
```

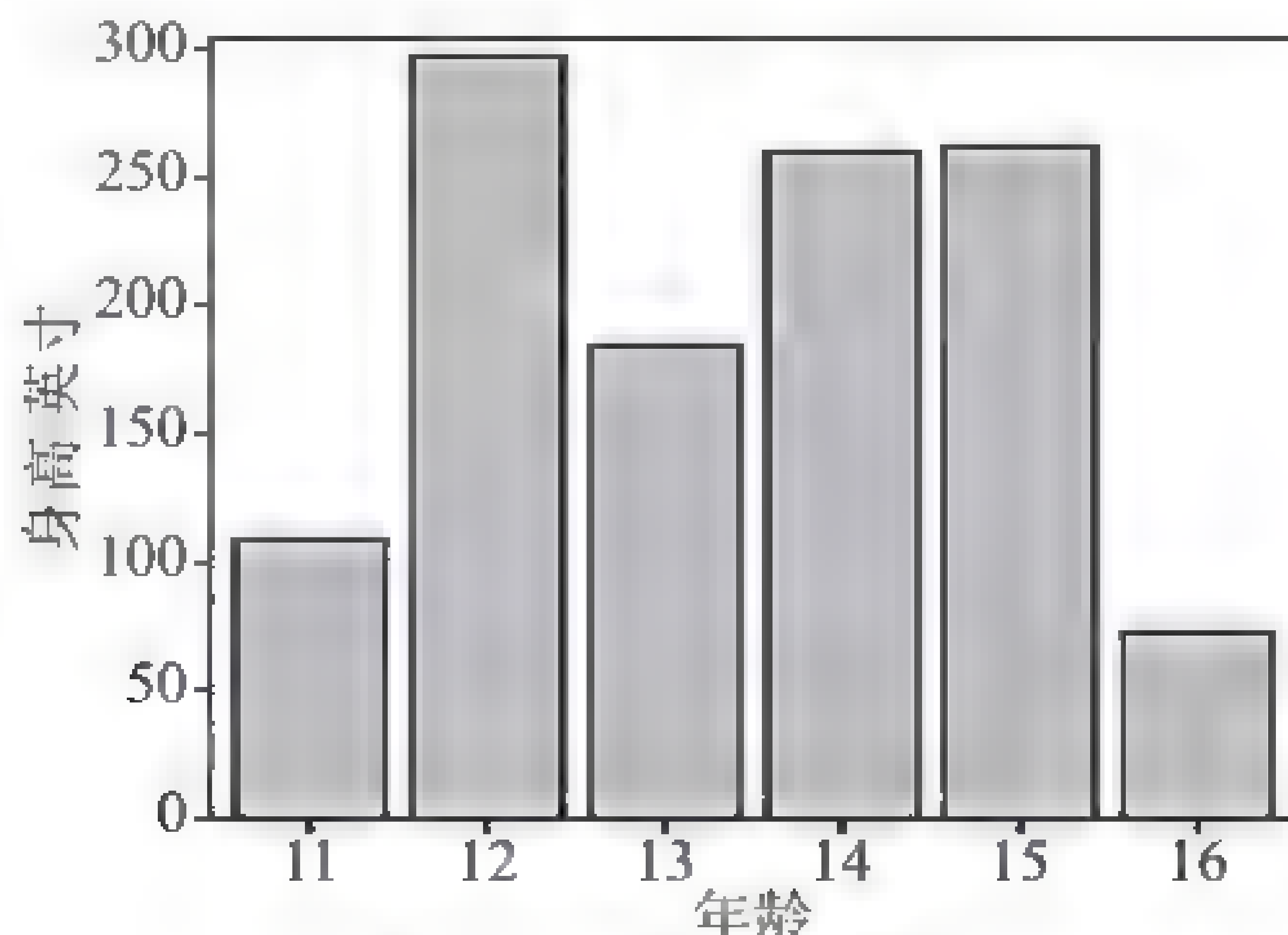


图 18-24 垂直条形图

为了在 Y 轴上显示平均值，而不是按照类别累计求和，需要在 vbar 语句上显式指定统计量为均值（见程序 18-28），输出结果如图 18-25 所示，此时纵坐标的数值为均值。

程序18-28　简单垂直条形图：统计量为均值

```
proc sgplot data=SASHELP.CLASS;
  vbar Age / response=Height stat=Mean;
  yaxis grid;
run;
```

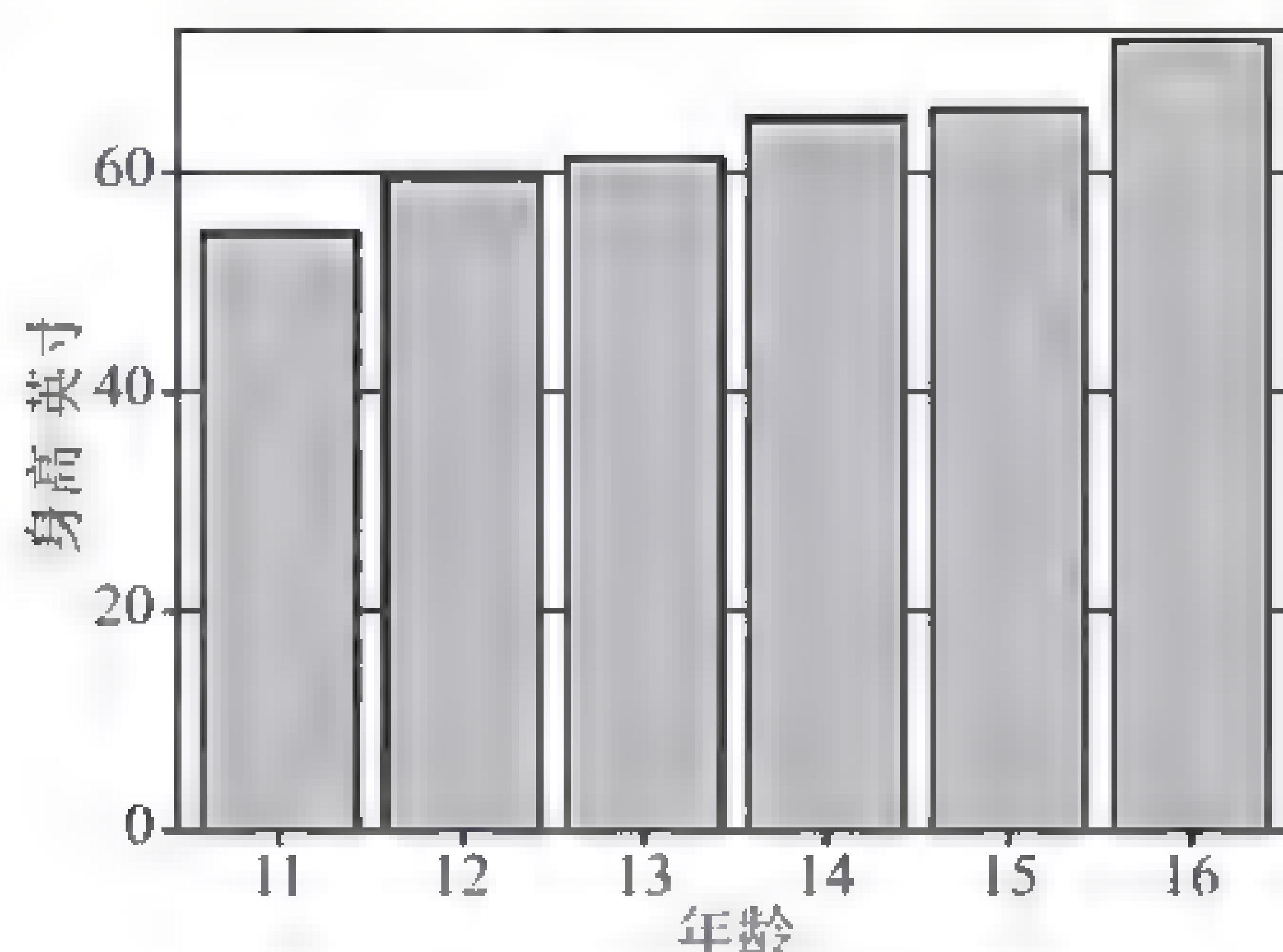


图 18-25　均值的垂直条形图

如果需要指定按照性别进行分组，还可以在 vbar 语句上指定分组变量。程序 18-29 的输出如图 18-26 所示。

程序18-29　分组堆叠垂直条形图

```
proc sgplot data=SASHELP.CLASS;
  vbar Age / response=Height stat=Mean group=Sex;
  yaxis grid;
run;
```

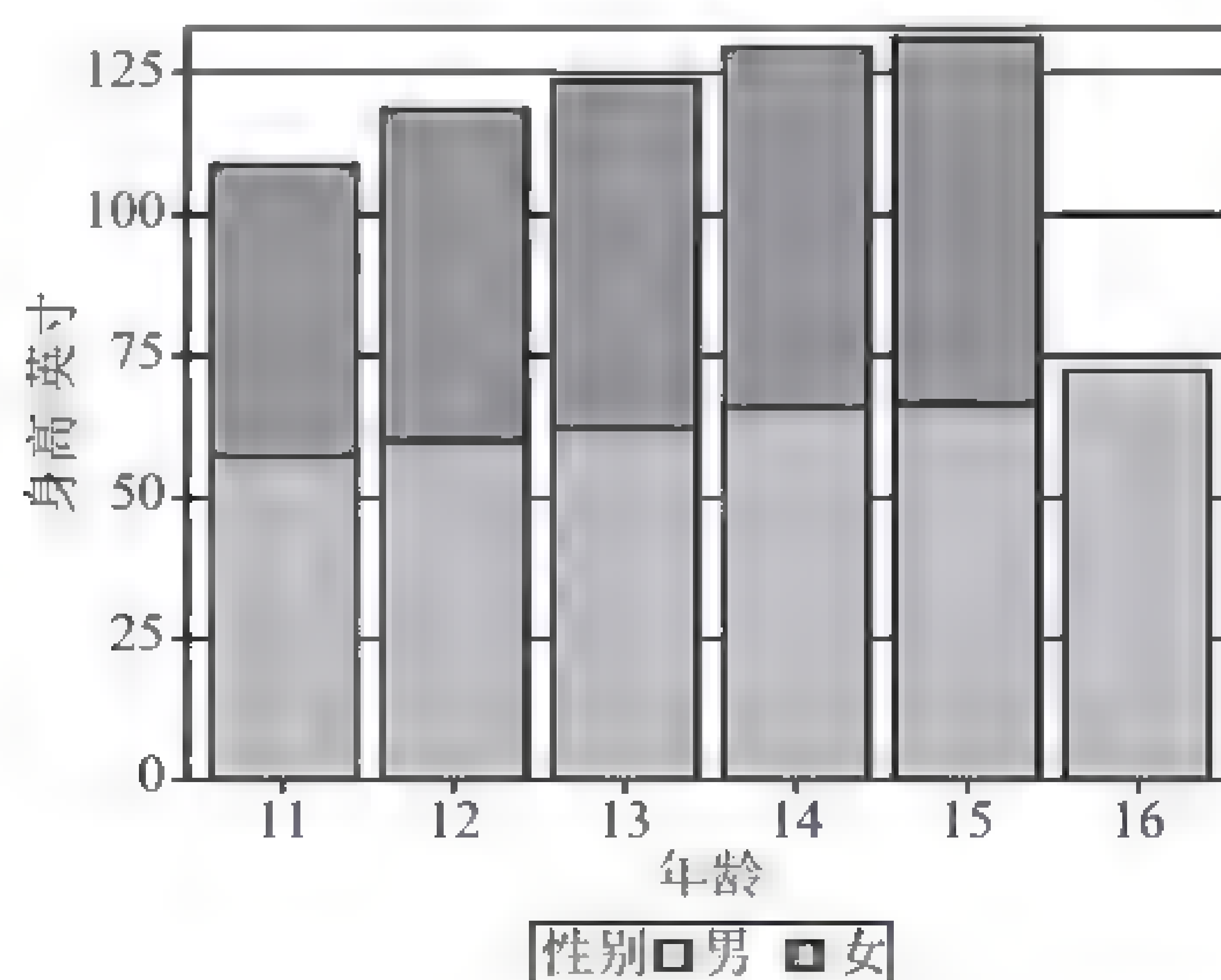


图 18-26　分组垂直条形图（堆叠）

为了直观对比两种性别的数据，我们可能不希望这种堆叠式的柱状图，而是能够对比差异的并列布局，则需要指定分组显示模式 GroupDisplay 为 Cluster（见程序 18-30 和图 18-27）。

```
程序18-30  分组并列垂直条形图
proc sgplot data SASHELP.CLASS;
  vbar Age / response=Height stat=Mean group=Sex groupdisplay=Cluster;
  yaxis grid;
run;
```

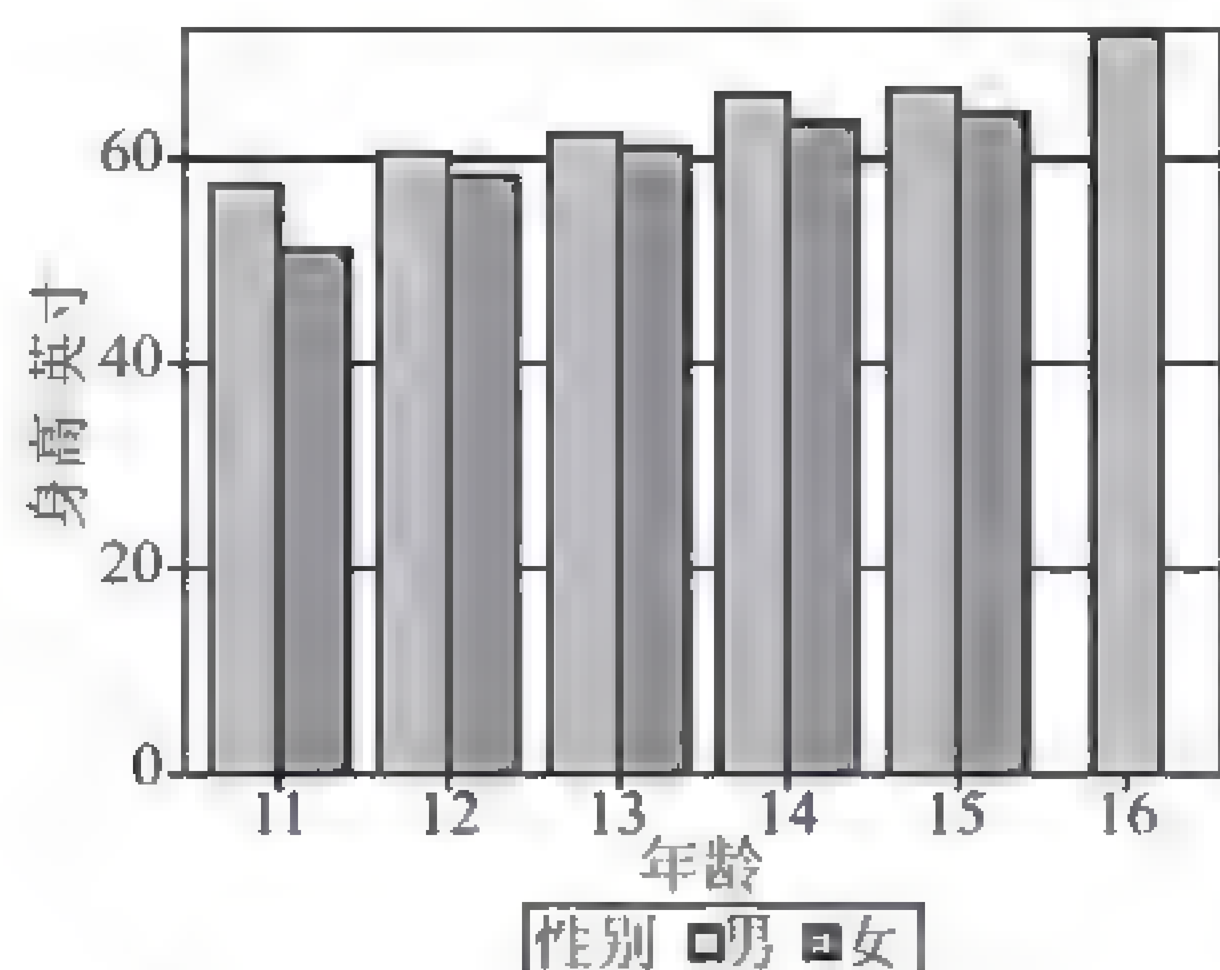


图 18-27 分组垂直条形图（并排）

如果希望根据性别绘制两个柱状图，而不是将它们放在同一柱状图中，则需要利用BY变量来分别生成对应性别的条形图，此时要求数据集包含BY变量，如果数据集没有预先根据该变量排序，则需要先排序后再调用SGPLOT过程步绘图。SAS大部分的图形图表过程步骤都支持利用BY变量生成多组图形（见程序18-31和图18-28）。

```
程序18-31  通过 BY 变量分组输出图形
proc sort data=SASHELP.CLASS out=SORT_CLASS;
  by Sex;/*先按照性别排序*/
run;
proc sgplot data=SORT_CLASS;
  by Sex;/*利用 BY 变量绘制一组图*/
  vbar Age / response=Height stat=Mean   ;
  yaxis grid;
run;
```

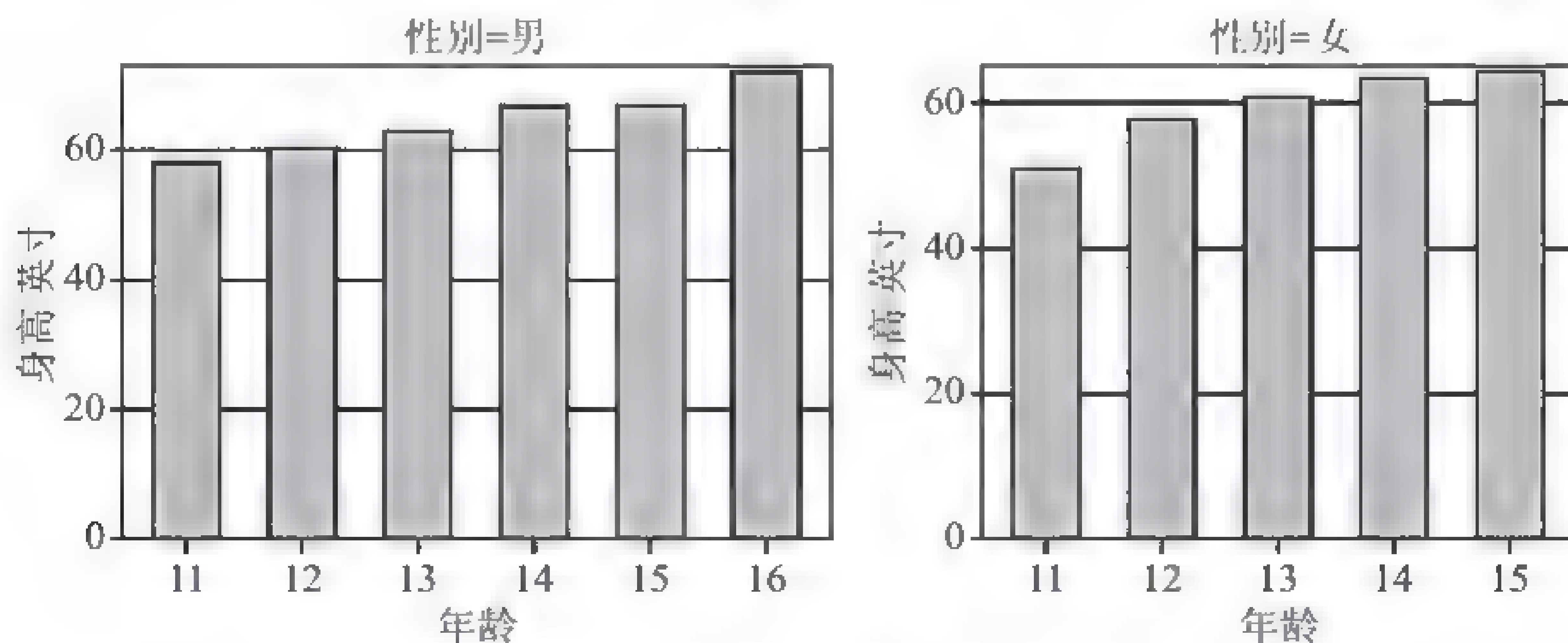


图 18-28 BY变量分别输出垂直条形图

SAS生成图形的时候还可以生成“交互式”图形，即用户可以在图上点击浏览或者跳转到特定报告或页面。在程序18-32生成的结果报表中，用户可以点击图形元素跳

转到指定的 URL 上，比如跳转到搜索引擎查找数据集中人名。这种模式需要启用 ODS GRAPHICS 特性，且数据集中有一个 URL 变量包含跳转的目标地址。

程序18-32　生成用户可交互的图表

```
ods graphics / reset imagemap;

data myclass;
  set sashelp.class;/*构造 URL 地址*/
  url='http://www.google.com/search?&q=' || trim(left(name));
run;
proc sgplot data=myclass;
  vbar Age / response=Height stat=Mean url=url;
  yaxis grid;
run;

ods graphics / reset;
```

柱状图和条形图适合对一个分类变量和一个数值型响应变量进行绘图。SGPLOT 还支持将多个兼容的图形绘制在同一个图中（见程序 18-33），比如可以把柱状图和线图结合起来形成条线图，它有两个垂直坐标轴：左边的 *Y* 轴和右边的 *Y*2 轴，其横坐标分类变量则相同（见图 18-29）。

程序18-33　双轴条线图的生成

```
proc sgplot data=SASHELP.CLASS nocycleattrs;
  yaxis grid min=0 max=80 ;
  vbar Age / response=Height stat=Mean ;

  y2axis min=0 max=200;
  vline Age / response=Weight stat=Mean y2axis
    lineattrs=(thickness=3);
  xaxis;
run;
```

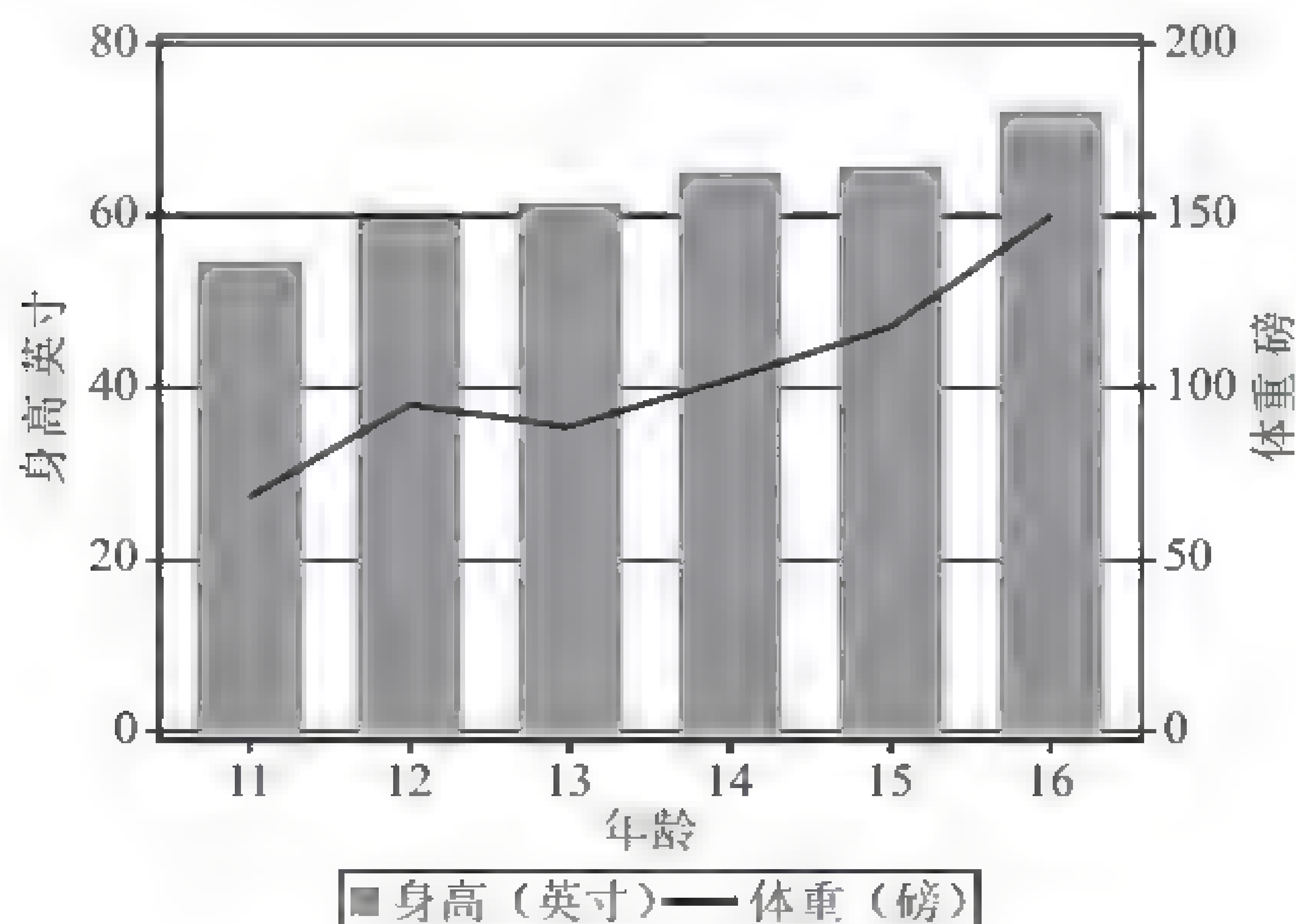


图 18-29　条线图

2. 直方图与盒形图

直方图是统计中最基础的图形，它可对某个变量进行分组统计。它与条形图是不同

的图形种类，直方图的纵坐标不需要指定响应变量，且其横坐标的各个柱之间是没有空隙，表示在数据区间上的分布情况（见程序 18-34 和图 18-30）。

程序18-34 直方图

```
proc sgplot data=SASHELP.CLASS;
  histogram Age;
  yaxis grid;
run;
```

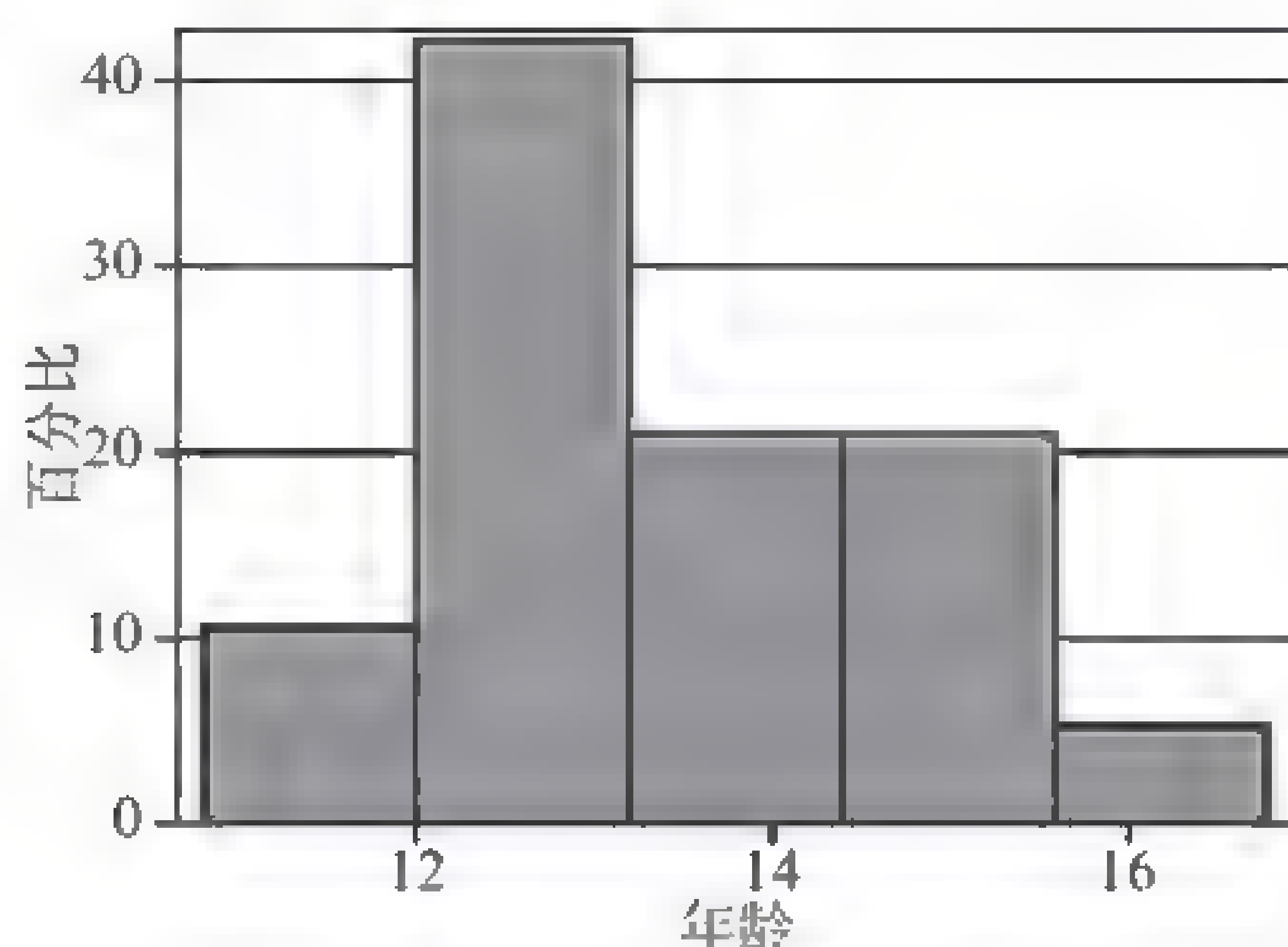


图 18-30 直方图

如果用户想要改变直方图的横坐标起点和宽度，可在 histogram 语句上指定选项，如下代码可设定从 10 开始，每个区间宽度为 1.2。

```
histogram Age / binwidth=1.2 binstart=10;
```

直方图数据也可通过编制频数表获得，比如通过 PROC FREQ 就能输出频数表。将下面代码（见程序 18-35）输出的表横过来看，其百分比就跟上面的直方图是一一对应（见图 18-31）。

程序18-35 频数表

```
proc freq data=sashelp.class;
  tables Age  / out=Freq_Age;
run;
proc print data=Freq_Age;run;
```

Age	频数	百分比	累积频数	累积百分比
11	2	10.53	2	10.53
12	5	26.32	7	36.84
13	3	15.79	10	52.63
14	4	21.05	14	73.68
15	4	21.05	18	94.74
16	1	5.26	19	100.00

图 18-31 频数表

当然，频数表可以使用多个变量进行交叉统计，此时输出结果中就包含频数、百分比、行百分比和列百分比等信息。

程序18-36 交叉统计表

```
proc freq data=sashelp.class;
  tables Age*Sex;
run;
```

频数 百分比 行百分比 列百分比	表 Age * Sex			
		Sex(性别)		
	Age(年龄)	男	女	合计
	11	1	1	2
		5.26	5.26	10.53
		50.00	50.00	
		10.00	11.11	
	12	3	2	5
		15.79	10.53	26.32
		60.00	40.00	
		30.00	22.22	
	13	1	2	3
		5.26	10.53	15.79
		33.33	66.67	
		10.00	22.22	
	14	2	2	4
		10.53	10.53	21.05
		50.00	50.00	
		20.00	22.22	
	15	2	2	4
		10.53	10.53	21.05
		50.00	50.00	
		20.00	22.22	
	16	1	0	1
		5.26	0.00	5.26
		100.00	0.00	
		10.00	0.00	
	合计	10	9	19
		52.63	47.37	100.00

图 18-32 交叉统计表（年龄 × 性别）

绘制直方图时可以同时绘制参考正态分布曲线和核密度估计曲线，分别用 DENSITY 语句指定。程序 18-37 展示了基于 SASHELP.CLASS 年龄变量的样本数据分布情况。

程序18-37 直方图和核密度估计曲线

```
proc sgplot data=SASHELP.CLASS;
  histogram Age  ;
  density Age;
  density Age / type=Kernel;
run;
```

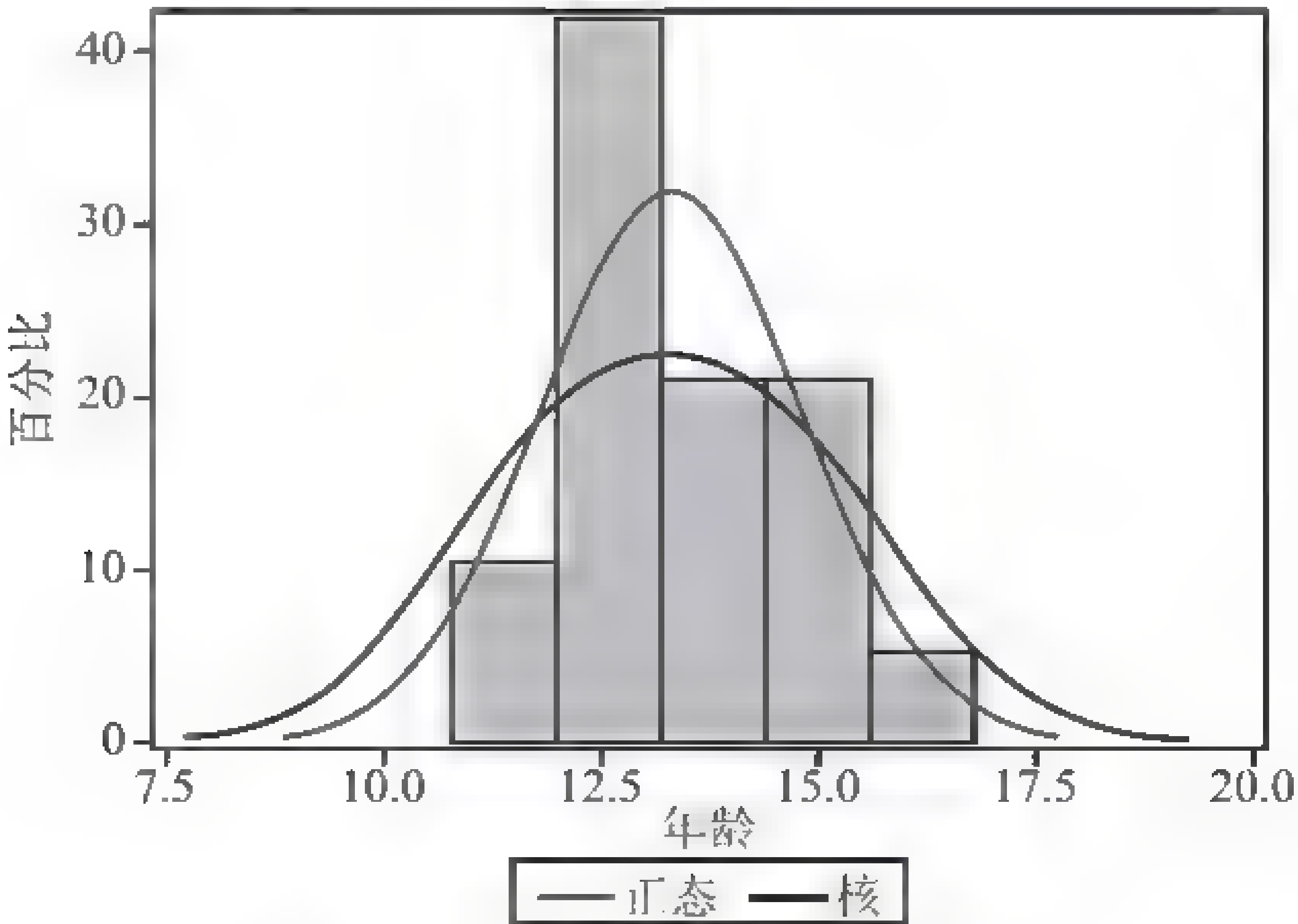


图 18-33 直方图与核密度估计曲线

盒形图可以展示某个样本的数据分布情况，它能在单个图上展现该变量的最小值、下四分位数、中位数、上四分位数和最大值，以及均值和异常值的情况。程序 18-38 展示了 Age 变量按照性别分组生成盒形图 18-34 的情况。从盒形图可看出女性的年龄中位数比男性的低。

程序18-38 分组盒形图揭示数据分布特征

```
proc sgplot data=SASHELP.CLASS;
  vbox age / group=sex fillattrs=(color=CXCAD5E5);
run;
```

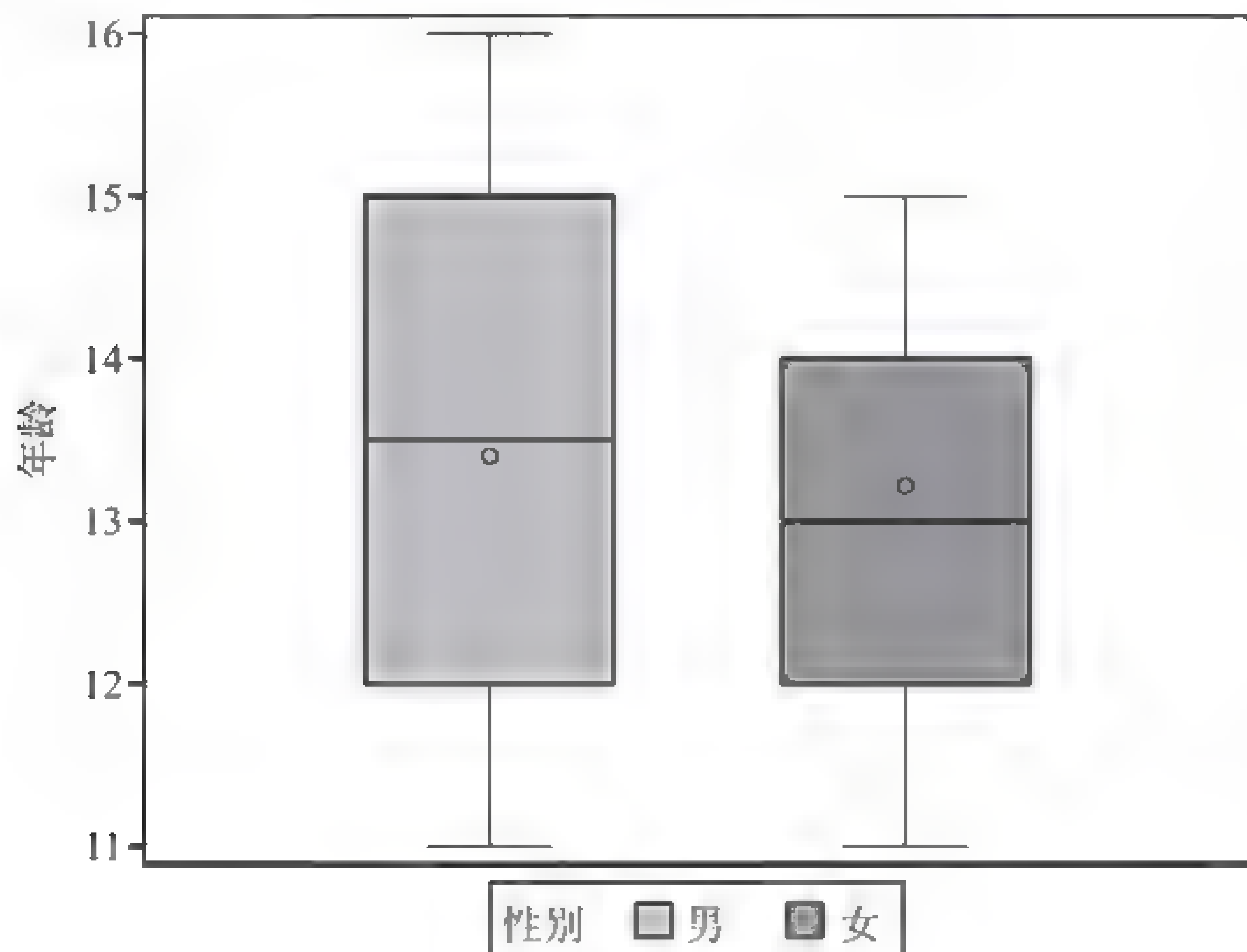


图 18-34 盒形图（分组）

3. 散点图及拟合

SGPLOT 支持散点图的绘制，同时它甚至可以直接进行回归拟合。下面的代码先用 scatter 语句按照性别不同用两种颜色绘制所有的点，然后用 reg 语句对体重和身高进行一次线性回归拟合（见程序 18-39）。如果想用二次方或更高次方程进行拟合，可在 reg 语句后指定 degree=2 或 3 即可。

程序18-39 散点图和线性回归拟合

```
proc sgplot data=SASHELP.CLASS;
  scatter x=Height y=Weight / group=sex datalabel=Name
    datalabelattrs=(size=7);
  reg x=Height y=Weight / nomarkers degree=1;
  xaxis grid;
  yaxis grid;
run;
```

程序运行的输出结果如图 18-35 所示。

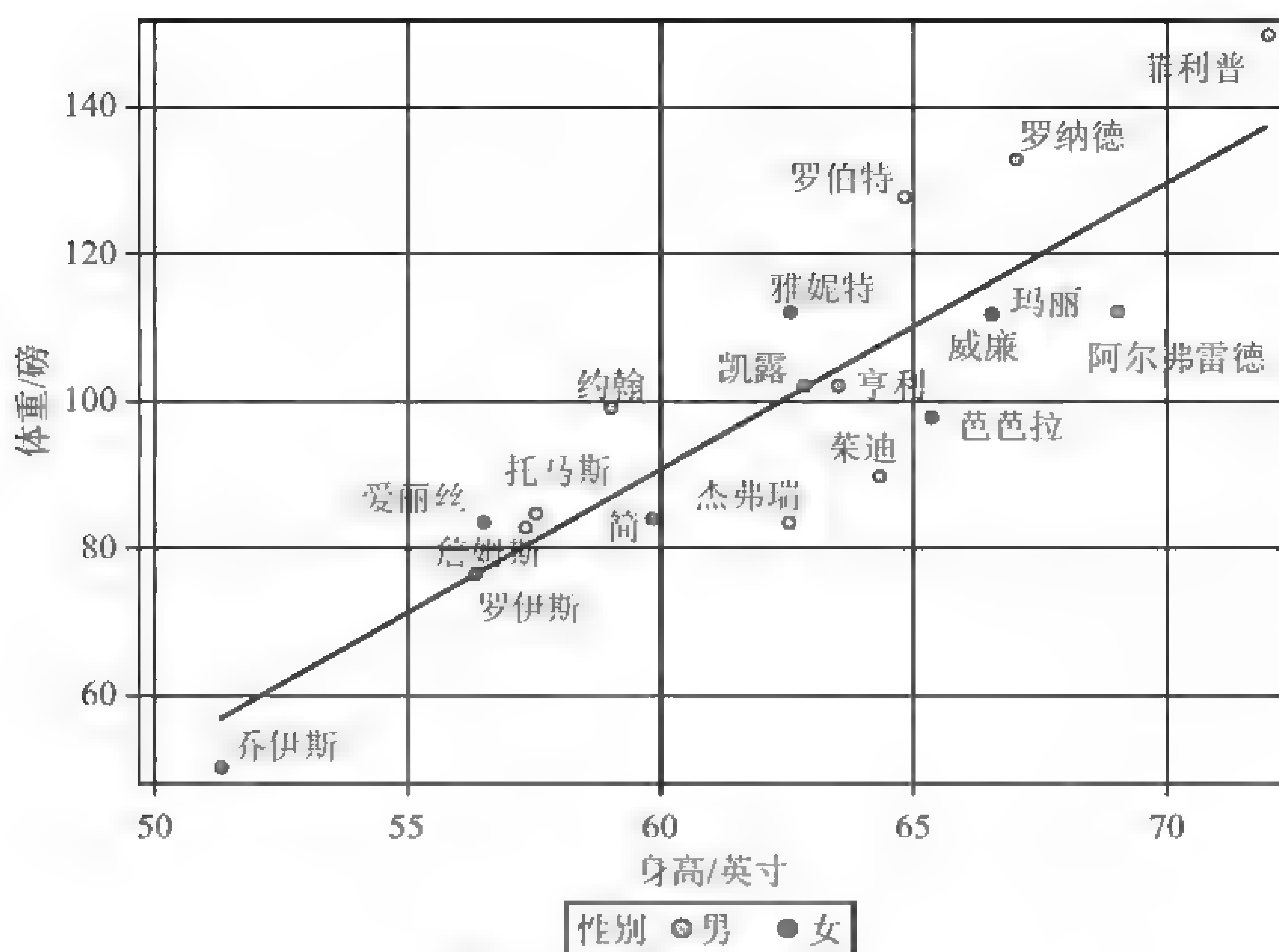


图 18-35　散点图及回归

如果要按性别分别对数据进行回归拟合，只需要在 reg 语句上增加一个 group=sex 属性，即可按照特定性别的数据进行分组线性回归（见图 18-36）。

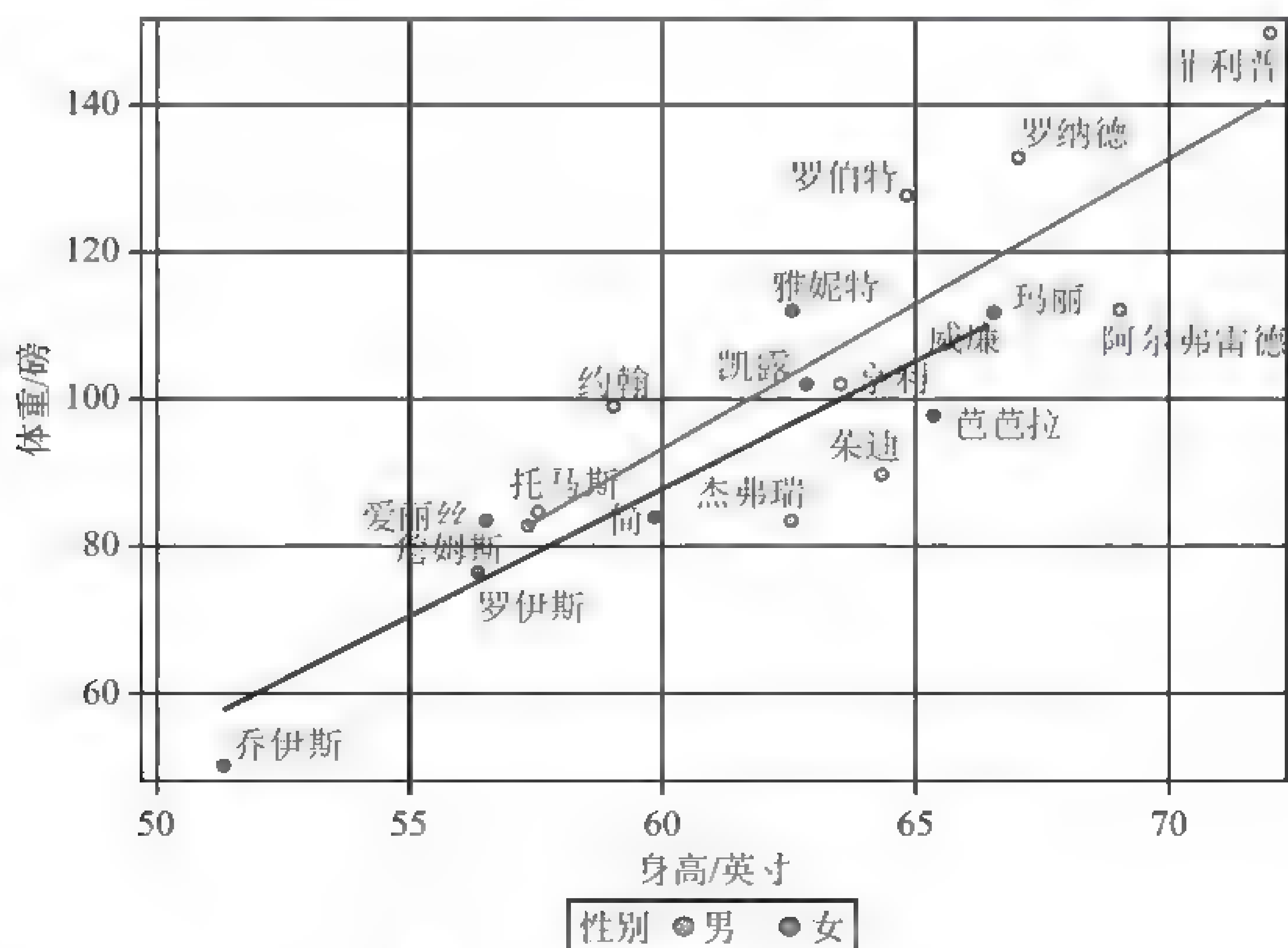


图 18-36　散点图及分组回归

此时如果想要把两图拆分，则需要把 SGPLOT 改成 SGPANEL 过程步，并通过 PANELBY 指定面板分组变量（见程序 18-40）。

程序18-40 SGPANEL分组绘制散点图和线性拟合

```
proc sgpanel data SASHELP.CLASS;
  panelby sex/ columns=2;

  scatter x=Height y=Weight / group=sex datalabel=Name datalabelattrs=(size=7);
  reg x=Height y=Weight / nomarkers;
run;
```

此时代码输出如图 18-37 所示。

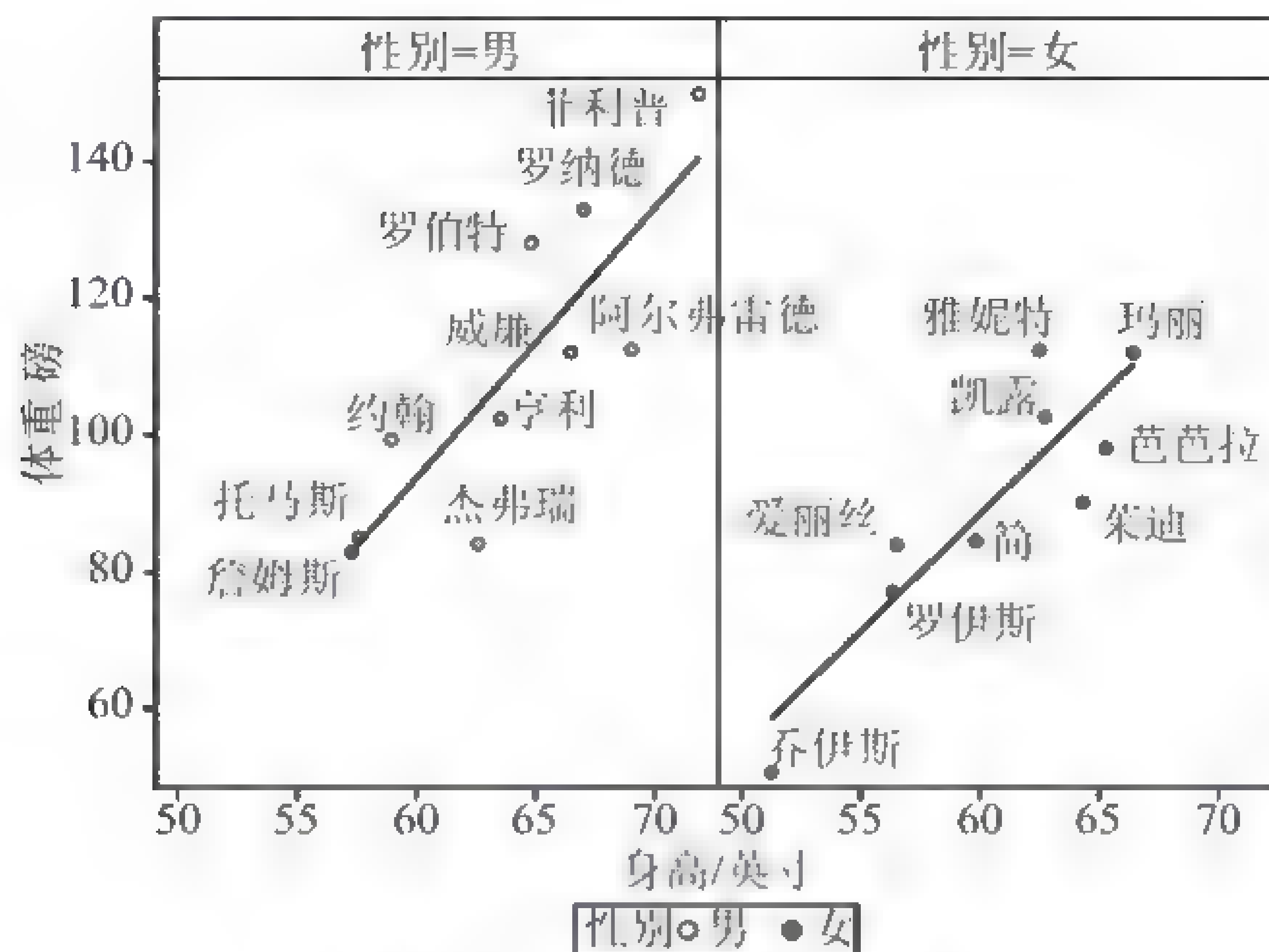


图 18-37 SGPANEL 拆分多图

在 Base SAS 中还有专门用于绘制散点图矩阵的过程步 SGSCATTER，比如程序 18-41 的 SAS 代码生成了两个散点图，纵坐标为体重和身高，横坐标为年龄，并且按照性别分组变量用不同的标记投点（见图 18-38）。

程序18-41 PROC SGSCATTER 绘制散点图

```
proc sgscatter data=sashelp.class;
  plot weight*age height*age / group=sex;
run;
```

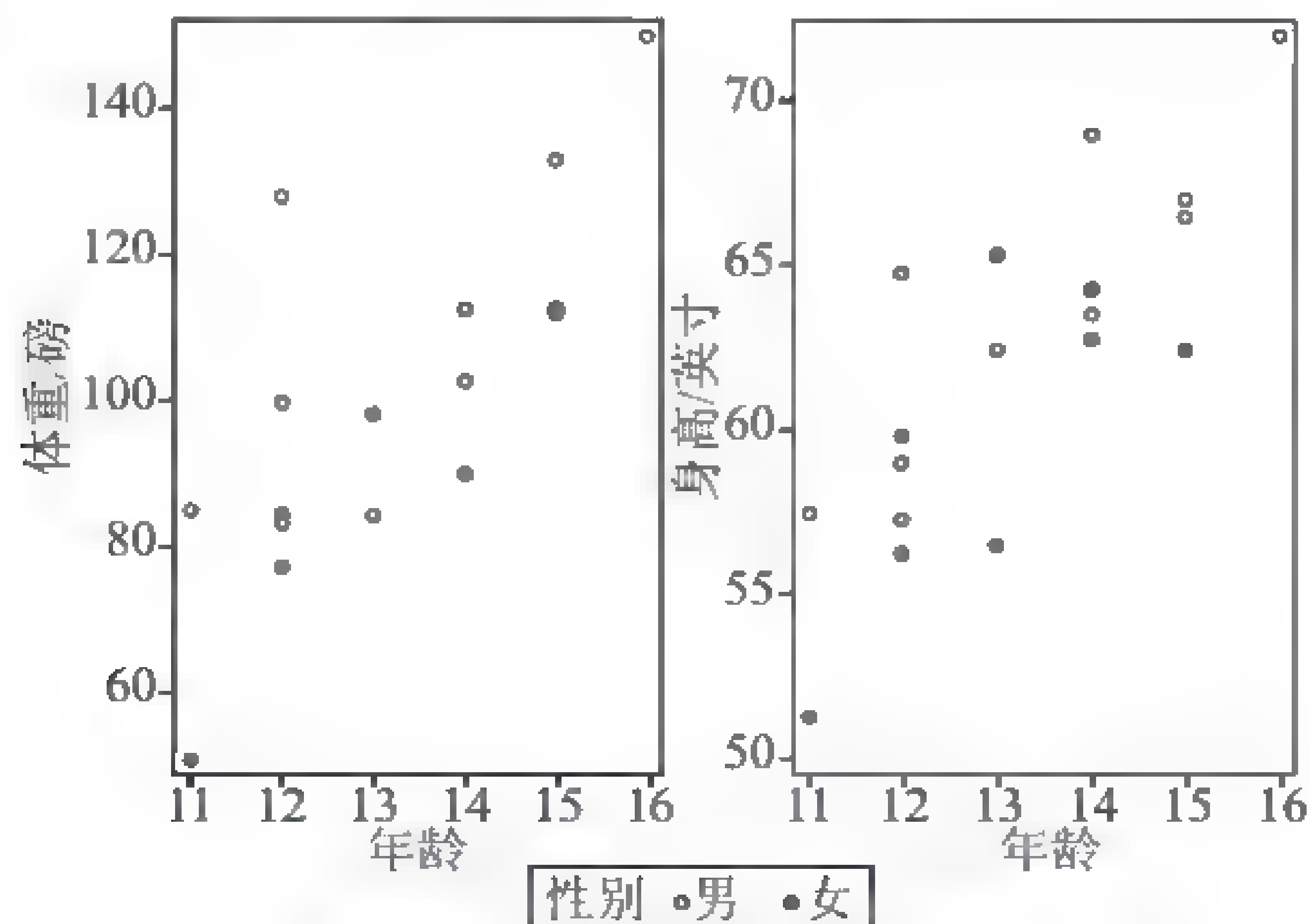


图 18-38 SGSCATTER 分组散点图

当分析的数据包含多个变量时，为了对比不同变量的直方图以及样品在两个变量空间的投点，可以用 SGCATTER 过程步绘制复合散点图和直方图矩阵（见程序 18-42 和图 18-39）。

程序18-42　绘制散点图矩阵，以及直方图／核密度曲线

```
proc sgscatter data=sashelp.class ;
  matrix age weight height / ellipse=(type=mean) diagonal=(histogram kernel);
run;
```

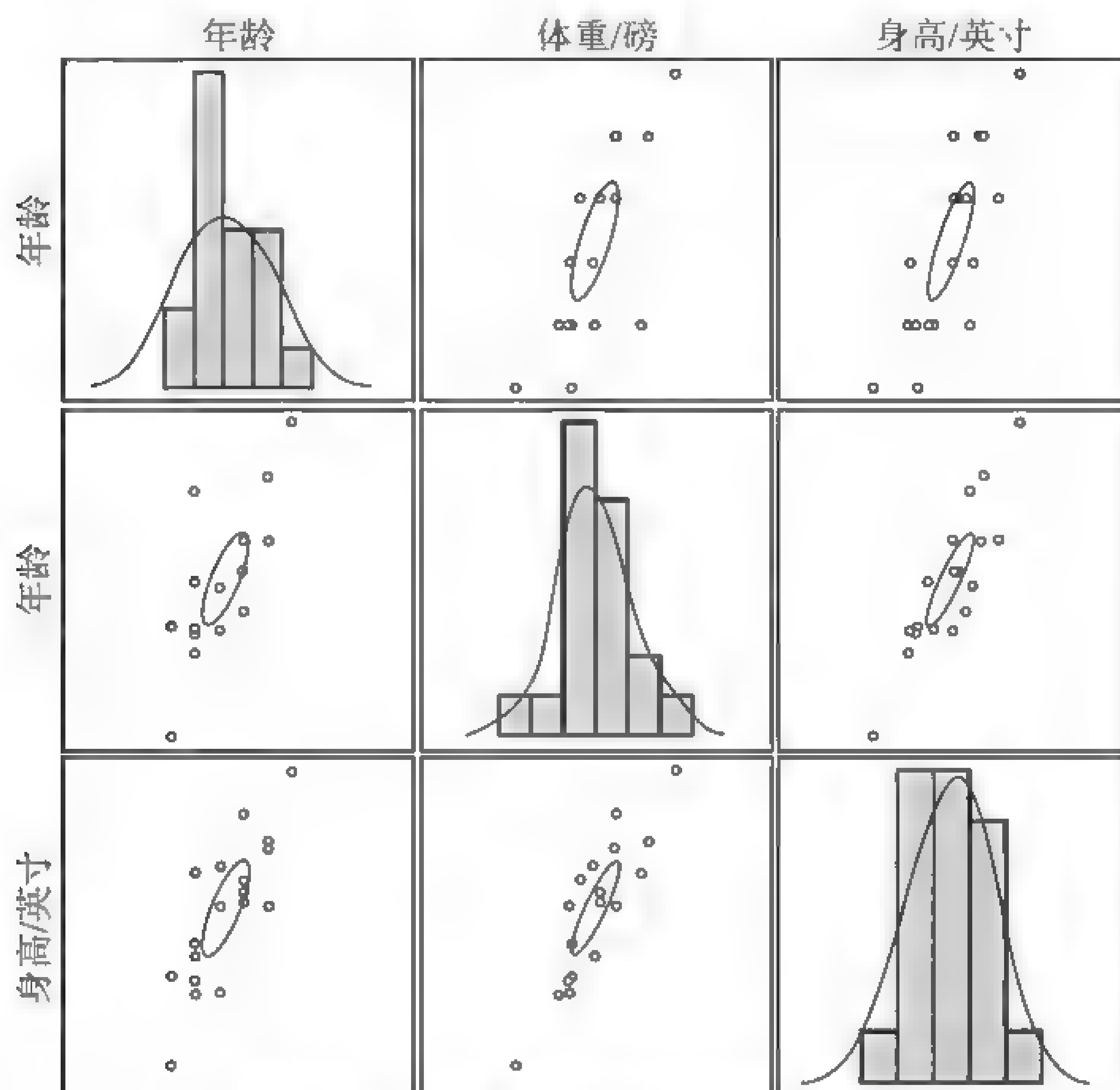


图 18-39　散点图矩阵与数据分布

4. 任意函数曲线——斐波那契螺旋线

在 SAS 中也可以绘制任何数学函数曲线，一般做法是先用函数生成数据序列，然后利用该数据进行绘图和分析即可。第 2 章的程序 2-42 利用正弦和余弦函数生成数据绘制了对应的函数曲线。程序 18-43 是作者编写的绘制斐波那契螺旋线（黄金螺旋），它展示了以斐波那契数为边长的正方形在平面上螺旋铺满整个平面的过程，而这些正方形的内切 1/4 圆弧构成了黄金螺旋线（详细步骤参见代码注释，结果如图 18-40 所示）。实际上，SAS 中可以非常方便绘制任何复杂的数学图形，包括等值线图、分形图、各种地图和彩色卫星地图等。

程序18-43　斐波那契螺旋线

```
data Fbnc;
  pi=constant("PI");
  do n=1 to 10;
    if n=1 or n=2 then x=1;
    else x= x1 + x2;
```

```
    x2=x1; x1=x;

    theta=(n+1) * pi/2;/*依次旋转 90 度*/
    if mod(n,4)=3 or mod(n,4)=0 then sign=-1;
    else sign=1;/*符号每次变换一次*/

    retain x0 y0;
    z=lag2(x);/*上上次的黄金分割数*/
    if n=1 OR n=2 then do;
      x0=1;
      y0=0;
    end;/*前两个中心点*/
    else do;/*交替变换中心点的 X/x 坐标*/
      if mod(n,2)=1 then x0=x0 + sign * z;
      else y0=y0 + sign * z;
    end;
    /*根据中心点计算内切 1/4 圆*/
    do t=theta to theta+pi/2 by 0.1;
      xx=x0 + x * cos(t);
      yy=y0 + x * sin(t);
      output;
    end;
    /*根据中心点，旋转前后两点绘制以斐波那契数为边的正方形*/
    xx=x0 + x * cos(theta+pi/2); yy=y0 + x * sin( theta+pi/2); output;
    xx=x0; yy=y0; output;
    xx=x0 + x * cos(theta); yy=y0 + x * sin(theta); output;
    xx=x0 + x * cos(theta); yy=y0 + x * sin(theta+pi/2); output;
    xx=x0 + x * cos(theta+pi/2); yy=y0 + x * sin( theta+pi/2); output;
  end;
  keep x xx yy;
  format x best32.;
run;
/*调用 SGPLOT 将所有点连起来，其中 width 和 aspect 用于保持正方形*/
ods graphics on / width=640pt border=off;
proc sgplot data=fbnc aspect=0.6 noautolegend ;
  series x=xx y=yy/group=x;
  xaxis display=none;
  yaxis display=none;
run;
```

程序运行后输出结果如图 18-40 所示。



图 18-40　斐波那契黄金数列

对于其他复合数学函数也可用类似的方法生成，思路都是先生成数据后调用 SGPLOT 绘制。但如果需要绘制三维图像，则需要用到 SAS/GRAPH 图形包中的一些

PROC 步来做，程序 18-44 的 13 行 SAS 代码可绘制三维函数图像，即 $z=f(x, y)$ 在三维空间的投影，结果如图 18-41 所示。

程序18-44　绘制三维函数图像

```
data mydata;
  PI=constant("PI");
  do x= -PI to PI by 0.05;
    do y= -PI to PI by 0.1;
      r=sqrt(x**2+y**2);
      z=cos(4*r)/r ;
      output;
    end;
  end;
run;
proc g3d data=mydata ;
  plot x*y=z   ;
run;
```

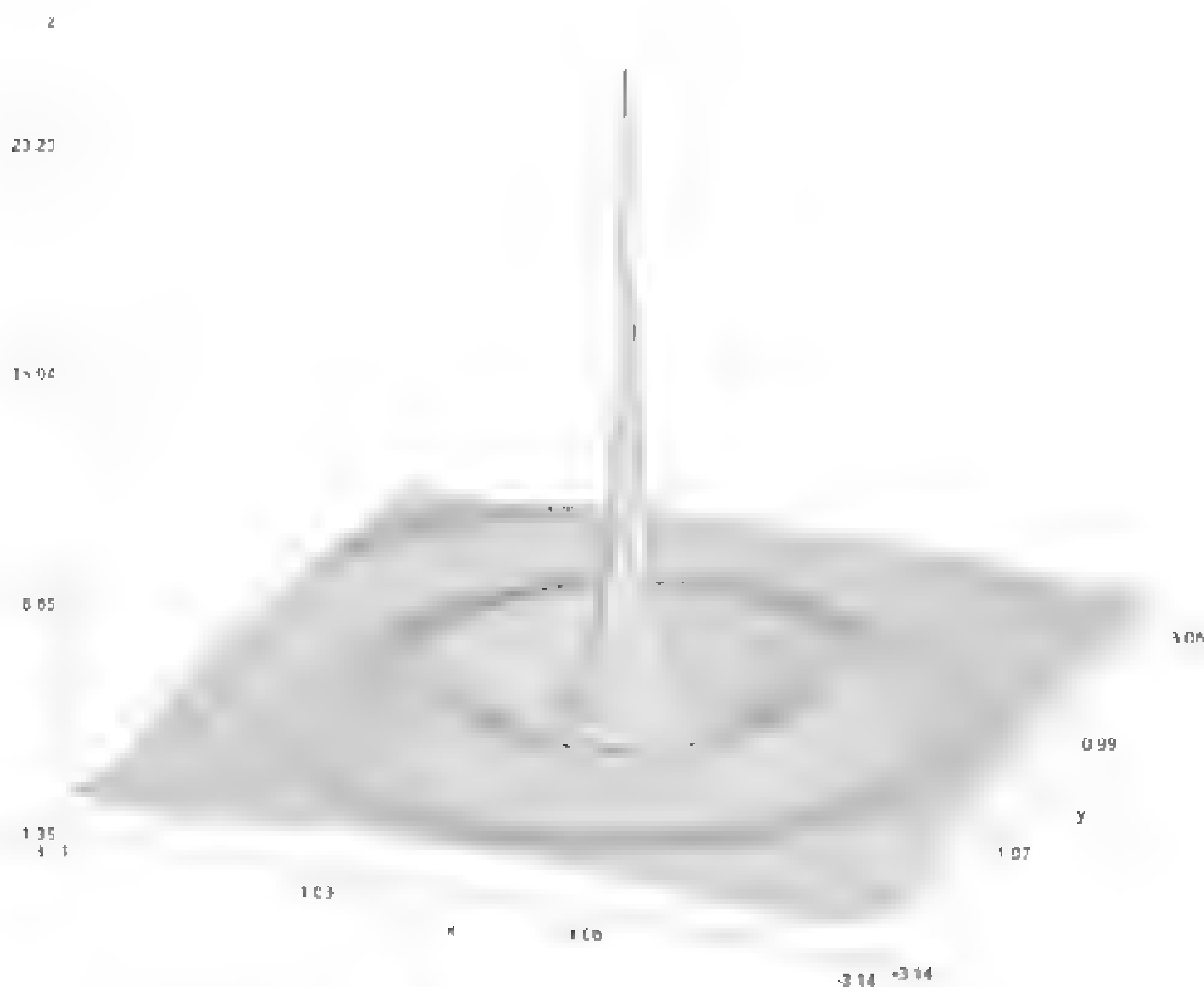


图 18-41　SAS 绘制三维图像

18.5　SAS 产品与过程步概览

经过 40 多年的发展，SAS 将各种数据分析方法和功能都进行了很好的封装和统一，形成了数百个严谨专业的统计分析 PROC 步，完成特定领域的分析功能。截至目前，SAS 33 个核心分析产品中包括了 425 个唯一命名的 PROC 步，其中 SAS/STAT 和 Base SAS 产品包含的 PROC 步最多，分别为 100 个和 95 个；其他如 SAS/ETS、SAS/GRAPH、SAS Visual Statistics 和 SAS VDMML 等产品也都包含 20 多个不等的过程步。

SAS 完整的 PROC 列表根据笔者的统计如图 18-42 所示。

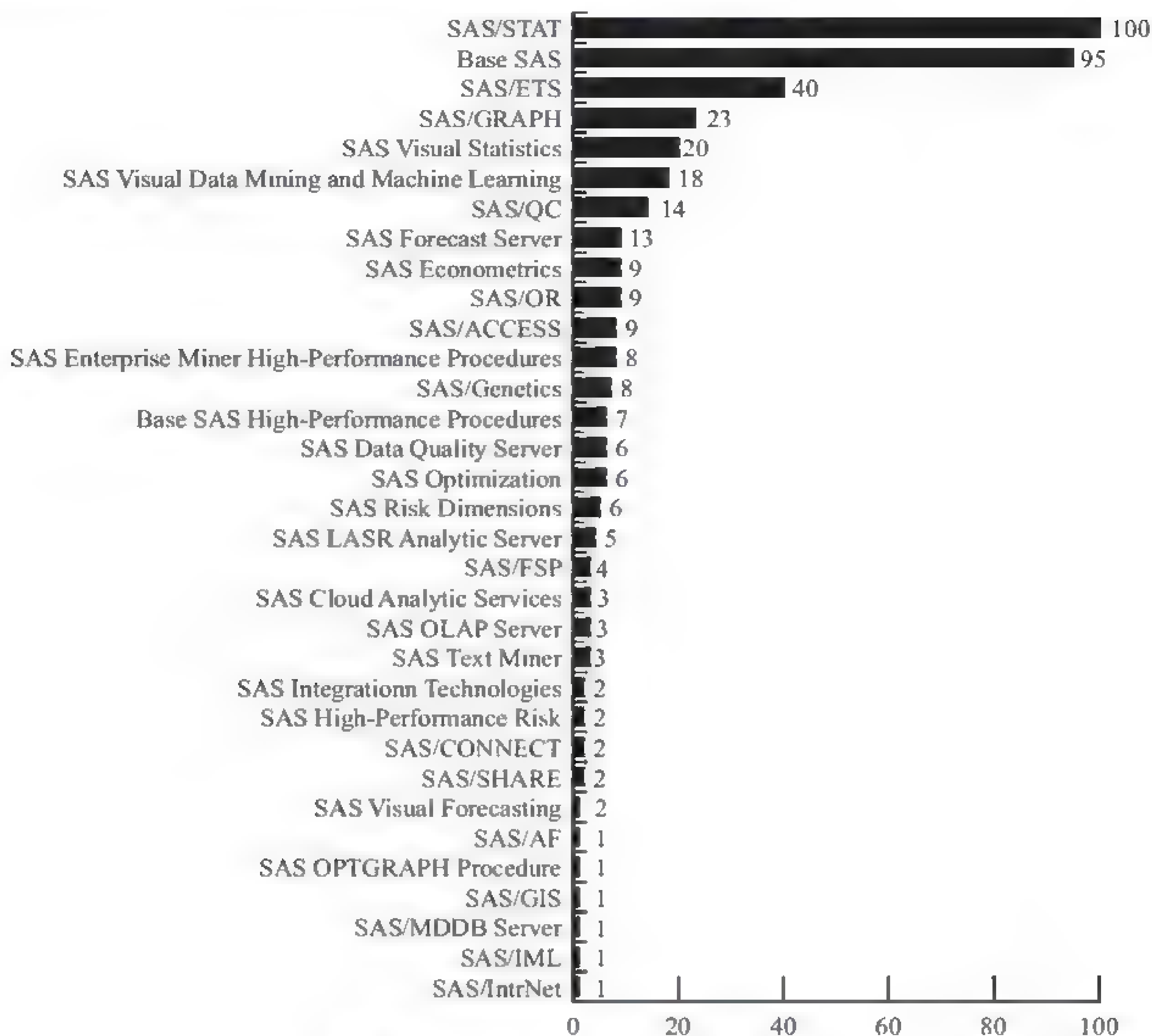


图 18-42　SAS 核心产品的过程步

33 个核心产品的结构关系图如图 18-43 所示，其中黑体部分为分析 PROC 过程步数量排名前 10 的 SAS 产品，而下划线部分为 SAS 的核心基础产品。

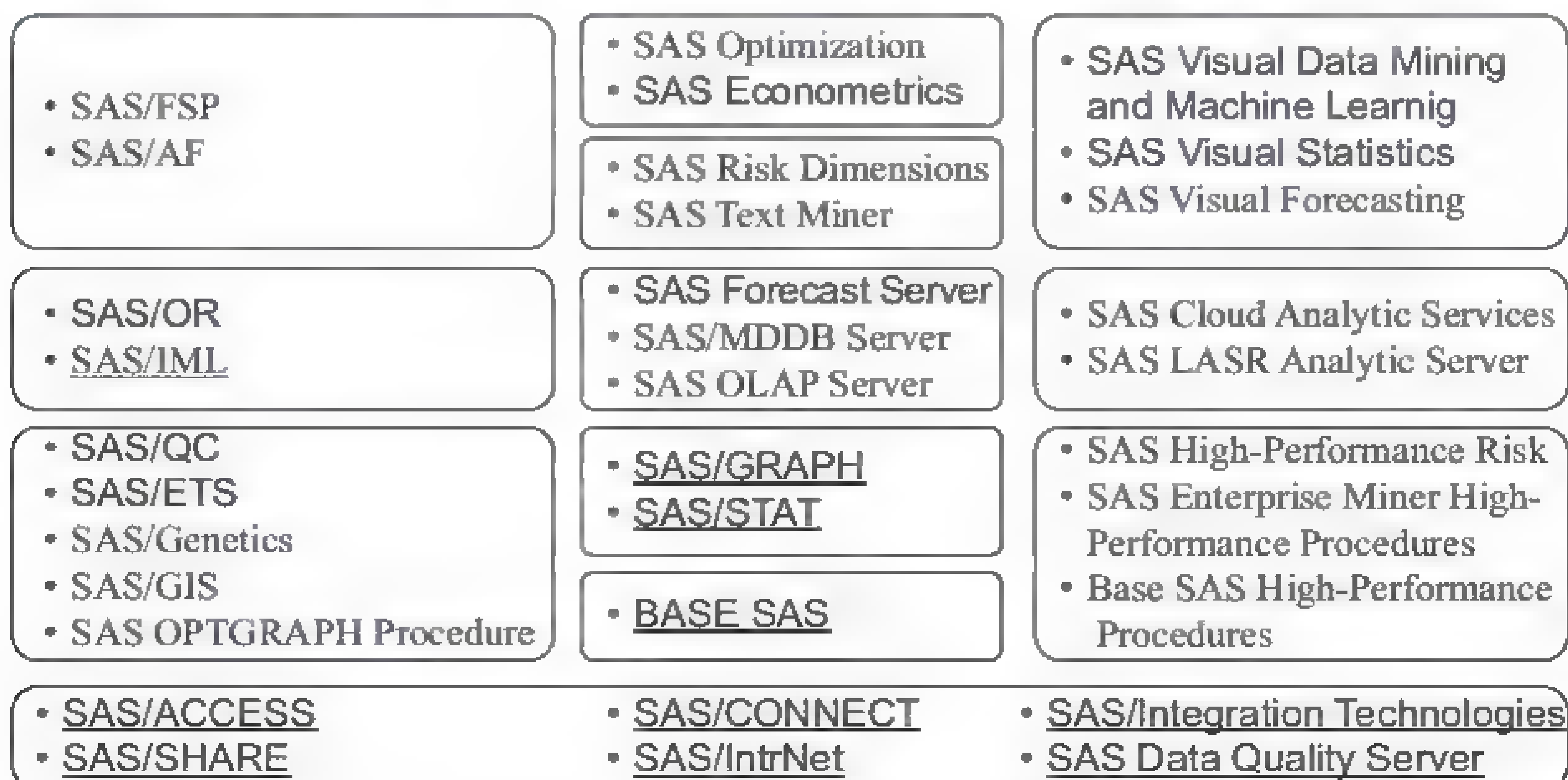


图 18-43　SAS 核心产品结构关系

18.5.1 SAS核心产品功能简介

下面列出了它们的主要功能和角色。理解它们对于帮助理解 SAS 产品家族各部分之间的相互关系具有重要意义。

（1）Base SAS 是 SAS 所有分析产品的基础和核心，它为数据分析提供可伸缩的集成软件环境，覆盖数据访问、变换和报告等功能。它包括 SAS 编程语言、数据操作、信息的存储和解析、描述性统计和报告编写等。同时，它也提供 SAS 宏来减少编程的时间和维护成本。具体形式上它包括数据步（DATA）语言、过程步（PROC）、DS2 语言、FedSQL 语言、SAS 宏、输出交付系统（ODS）和 ODS 图形等。

最新的 Base SAS 产品还提供访问 SAS Cloud Analytics Service (CAS) 和 SAS Viya 平台的能力。在支持 CAS 服务器的 SAS 运行会话中，除了支持传统的 DATA 步和 PROC 步外，还可通过 PROC CAS 用 CASL 语言编写 CAS 动作把数据加载到 CAS 服务器上，或者将 CAS 数据表保存到分布式运行环境上。CAS 服务器上可运行 SAS 新一代的 Viya 分析过程步以及 CAS 服务器管理过程步。对于提交的 DATA 步语言，如果它包含不能运行在 CAS 服务器上的代码，该数据步会智能地运行在 SAS 运行环境中；有些基础过程步如 MEANS、SUMMARY、REPORT、TABULATE、TRANSPOSE 和 COPY 支持在 CAS 服务器上运行。CAS 服务器也支持运行 PROC DS2 代码，并支持 PROC FEDSQL 过程步提交 SQL 语言到 CAS 服务器上执行查询和表关联功能。

（2）SAS/GRAPH – 为企业提供全方位的商业数据可视化呈现能力，灵活而精细的图形图表控制和结果呈现能力，为企业用户的决策过程提供丰富而有意义的视觉体验。SAS/GRAPH 提供几乎所有的图形图表绘制，也提供强大的自定义功能。

（3）SAS/STAT – 提供企业所需的各种数据分析统计和专业领域的统计功能；从 1972 年首次发布 SAS/STAT 72 版本至今已经发展了 45 年，是全球首屈一指的全功能商业统计包。2017 年 9 月最新发布的 14.3 (SAS 9.4M5) 新增 CAUSALMED 过程步，并对一系列过程步进行增强，包括 GAMPL、FREQ、IRT、NLMIXED、MCMC、PHREG、QUANTREG、QUANTSELECT 和 TTEST。每个过程步都包含了极为丰富的参数和选项控制，提供一致或类似的访问接口。

（4）SAS/ACCESS 通过实现 DBMS 数据库厂商的 API 或语言接口，或者通过 DBMS 厂商或第三方的 ODBC 驱动实现在 SAS 中对各种数据库 / 数据源的访问。它通过 3 个核心 PROC 步来实现广泛的数据访问：IMPORT/EXPORT/METALIB。它提供 25 种数据库访问接口以及元数据访问接口，包括通用的 ODBC, OLE DB 接口以及 Oracle、Microsoft SQL Server、MySQL、Hadoop、Teradata、Greenplum 等主流数据库厂商。

（5）SAS SHARE 为 SAS 提供多用户的客户服务器运行环境，支持多用户的服务器能为本地用户或远程用户在企业环境下提供对 SAS 数据的并发更新。服务器在为

远程用户读取数据时提供低开销的网络连接，并支持对第三方 DBMS 产品的联合数据访问。

（6）SAS/CONNECT　将网络中不同操作系统的计算机连接起来，有效分发计算负载到网络中的各计算机来达到并行处理的效果。

（7）SAS/IntrNet　用来开发企业 Web 应用开发接口，访问后台的 SAS 产品功能。其核心部件 “应用分发器”是基于 CGI 技术开发的 Web 网关，它可以接受来自客户端浏览器的调用请求，对后台等待的 SAS 会话进行调用和返回。

（8）SAS Integration Technologies　提供创建以访问 SAS 服务器功能作为后台的分布式企业应用所需的各种服务，包括应用消息、BI WEB 服务、目录服务、JMX、负载均衡、单点登录、发布框架、存储过程和 Web 基础设施平台等企业应用基础服务。

（9）SAS/IML – 为程序员，统计学家和研究人员在 SAS 中使用强大灵活的交互式矩阵编程语言 IML 提供可能，允许在 SAS 语言中操纵数据和统计分析，然后用 SAS/IML 语言做更加专业化的分析和探索。

（10）Base SAS High Performance Procedures – 是 SAS 专门为高性能分布式分析服务器新开发的基础过程步，包括 HPBIN, HPCORR, HPDMDB, HPDS2, HPIMPUTE, HPSAMPLE, HPSUMMARY 7 个过程步。这些高性能分析过程步可以运行在单机环境模式，也可以运行在分布式模式中，但后者要求有相应的高性能分析产品授权。

（11）SAS LASR Analytic Server – 是 SAS 大数据时代的内存分析服务器，它为从分布式计算环境中加载到内存中的数据提供安全的多用户并发访问。SAS LASR Analytic Server 分布式环境中各个节点内存对客户端而言就像是一块连续的内存，磁盘上的数据被持久化到分布式内存后，所有的数据访问都是内存中进行，从而能够对数据提供迅捷的分析操作。通过避免反复的数据加载和卸载操作，它能够对海量数据提供秒级访问。LASR 服务器上的数据是只读且无状态的，LASR 服务器是一个内存分析平台，而不是一个分布式内存数据库。

（12）SAS Cloud Analytic Services – 是 SAS 2016 年推出的支持部署在云端的高性能分布式数据管理和分析运行环境。SAS 是全球分布式并行计算领域的先驱和领导者，而 CAS 云分析服务是 SAS 从 1972 年至今，经过 MVA、TK、IN-DATABASE & HPA、LASR 等各种计算架构演进后的最新发明，它是数据模式和分析逻辑达到最佳平衡的分布式内存计算环境。SAS CAS 云分析服务支持对亿行数据在亚秒级的实时数据分析。

SAS 中各核心产品的 PROC 按照字母顺序排序如表 18-3 所列。黑体表示支持 UNIX 平台，斜体表示支持 Windows 平台，下划线表示支持 z/OS, 否则表示支持任意操作系统平台。

表 18-3 核心产品过程步索引

基础模块	Base SAS（95） APPEND AUTHLIB CALENDAR *CATALOG* CDISC CHART CIMPORT COMPARE *CONTENTS* *CONVERT* COPY CORR *CPORT* *DATASETS* DATEKEYS DBCSTAB DELETE DISPLAY DOCUMENT DS2 DSTODS2 EXPLODE EXPORT FCMP FEDSQL FMTC2ITM FONTREG FORMAT FORMS FREQ FSLIST GROOVY HADOOP HDMD HTTP IMPORT INFOMAPS ITEMS JAVAINFO JSON LOCALEDATA LUA MEANS METADATA METALIB METAOPERATE MIGRATE ODSLIST ODSTABLE ODSTEXT *OPTIONS* OPTLOAD OPTSAVE PDS PDSCOPY PLOT *PMENU* PRESENV PRINT PRINTTO *PROTO* PRTDEF PRTEXP PWENCODE QDEVICE RANK REGISTRY RELEASE REPORT S3 SCAPROC SCOREACCEL SGDESIGN SGMAP SGPANEL SGPLOT SGRENDER SGSCATTER SOAP *SORT* SOURCE SQL SQOOP STANDARD STREAM SUMMARY TABULATE TAPECOPY TAPELABEL TEMPLATE: TIMEPLOT TRANSPOSE TRANTAB UNIVARIATE XSL
统计分析模块	SAS/STAT (100) ACECLUS ADAPTIVEREG ANOVA BCHOICE BOXPLOT CALIS CANCORR CANDISC CATMOD CAUSALMED CAUSALTRT CLUSTER CORRESP DISCRIM DISTANCE FACTOR FASTCLUS FMM GAM GAMPL GEE GENMOD GLIMMIX GLM GLMMOD GLMPOWER GLMSELECT HPCANDISC HPFMM HPGENSELECT HPLMIXED HPLOGISTIC HPMIXED HPNLMOD HPPLS HPPRINCOMP HPQUANTSELECT HPREG HPSPLIT ICLIFETEST ICPHREG INBREED IRT KDE KRIGE2D LATTICE LIFEREG LIFETEST LOESS LOGISTIC MCMC MDS MI MIANALYZE MIXED MODECLUS MULTTEST NESTED NLIN NLMIXED NPAR1WAY ORTHOREG PHREG PLAN PLM PLS POWER PRINCOMP PRINQUAL PROBIT PSMATCH QUANTLIFE QUANTREG QUANTSELECT REG ROBUSTREG RSREG SCORE SEQDESIGN SEQTEST SIM2D SIMNORMAL SPP STDIZE STDRATE STEPDISC SURVEYFREQ SURVEYIMPUTE SURVEYLOGISTIC SURVEYMEANS SURVEYPHREG SURVEYREG SURVEYSELECT TPSPLINE TRANSREG TREE TTEST VARCLUS VARCOMP VARIOGRAM
图形图表	SAS/GRAPH（23） G3D G3GRID GANNO GAREABAR GBARLINE GCHART GCONTOUR GDEVICE GEOCODE GFONT GINSIDE GKPI GMAP GOPTIONS GPLOT GPROJECT GRADAR GREDUCE GREMOVE GREPLAY GSLIDE GTILE MAPIMPORT
多维与预测	SAS Forecast Server (13): HPF HPFARIMASPEC HPFDIAGNOSE HPFENGINE HPFESMSPEC HPFEVENTS HPFEXMSPEC HPFIDMSPEC HPFRECONCILE HPFREPOSITORY HPFSELECT HPFTEMPRECON HPFUCMSPEC SAS/MDDB Server (1): MDDB SAS OLAP Server (3) : OLAP OLAPCONTENTS OLAPOPERATE
风险与文本	SAS Risk Dimensions (6): AGGREGATION COMPILE RDC RDPOOL RDSEC RISK SAS Text Miner (3): HPBOOLRULE HPTMINE HPTMSCORE

（续表）

优化与计量经济	SAS Optimization (6): CLP OPTLP OPTMILP OPTMODEL OPTNETWORK OPTQP SAS Econometrics (9): CCDM CCOPULA CNTSELECT CPANEL CQLIM CSPATIALREG HMM SEVSELECT TSMODEL
数据访问与集成技术	SAS/ACCESS (9): ACCESS CALLRFC CV2VIEW DB2EXT DB2UTIL DBF DBLOAD DIF QUEST SAS/SHARE (2): OPERATE SERVER SAS/CONNECT (2): DOWNLOAD UPLOAD SAS/IntrNet (1): APPSRV SAS Integration Technologies (2): IOMOPERATE STP SAS Data Quality Server (6): DMSRVADM DMSRVDATASVC DMSRVPROCESSSVC DQLOCLST DQMATCH DQSCHEME
应 用 开 发	SAS/FSP (4): FSBROWSE FSEDIT FSLETTER FSVIEW SAS/AF (1): BUILD
质量控制与 ETS 模块	SAS/OR（9）: BOM CPM DTREE GA GANTT NETDRAW OPTLSO OPTNET PM SAS/IML (1): IML SAS/QC (14): ANOM CAPABILITY CUSUM FACTEX ISHIKAWA MACONTROL MVPDIAGNOSE MVPMODEL MVPMONITOR OPTEX PARETO RAREEVENTS RELIABILITY SHEWHART SAS/ETS (40) ARIMA AUTOREG COMPUTAB COPULA COUNTREG DATASOURCE ENTROPY ESM EXPAND FORECAST HPCDM HPCOPULA HPCOUNTREG HPPANEL HPQLIM HPSEVERITY LOAN MDC MODEL PANEL PDLREG QLIM SEVERITY SIMILARITY SIMLIN SPATIALREG SPECTRA SSM STATESPACE SYSLIN TIMEDATA TIMEID TIMESERIES TMODEL TSCSREG UCM VARMAX X11 X12 X13 SAS/Genetics (7): ALLELE BTL CASECONTROL FAMILY GENESELECT HAPLOTYPE HTSNP PSMOOTH SAS/GIS (1): GIS SAS OPTGRAPH Procedure (1): OPTGRAPH
可视化分析与机器学习	SAS Visual Data Mining and Machine Learning (18) ASTORE BNET BOOLRULE FACTMAC FASTKNN FISM FOREST GRADBOOST GVARCLUS MBANALYSIS MWPCA NETWORK NNET RPCA SVDD SVMACHINE TEXTMINE TMSCORE SAS Visual Statistics (20) ASSESS BINNING CARDINALITY CORRELATION FREQTAB GAMMOD GENSELECT KCLUS LOGSELECT NLMOD PARTITION PCA PHSELECT PLSMOD QTRSELECT REGSELECT SPC TREESPLIT VARIMPUTE VARREDUCE SAS Visual Forecasting (2): TSINFO TSRECONCILE

（续表）

基础服务	SAS Cloud Analytic Services (3): CAS CASUTIL MDSUMMARY SAS LASR Analytic Server (5): IMSTAT IMXFER LASR RECOMMEND VASMP
高性能过程步	SAS High-Performance Risk (2): HPEXPORT HPRISK SAS Enterprise Miner High-Performance Procedures (8): HP4SCORE HPBNET HPCLUS HPDECIDE HPFOREST HPNEURAL HPREDUCE HPSVM Base SAS High-Performance Procedures (7): HPBIN HPCORR HPDMDB HPDS2 HPIMPUTE HPSAMPLE HPSUMMARY

18.5.2　Base SAS 过程步速查

本节内容为笔者根据数据分析中使用频率而整理的全部 Base SAS 过程步说明，共 95 个，供读者快速参考之用。具体每个 PROC 都包含大量的参数和语句，使用时还需查看联机帮助或手册进行。Base SAS 过程步覆盖的功能包括：基础操作、数据导入导出、功能扩展、系统集成、图形图表、基础分析、输出管理、环境管理、元数据管理和 z/OS 大型机特定功能。从系统功能扩展角度看，Base SAS 各模块之间的结构关系如图 18-44 所示。

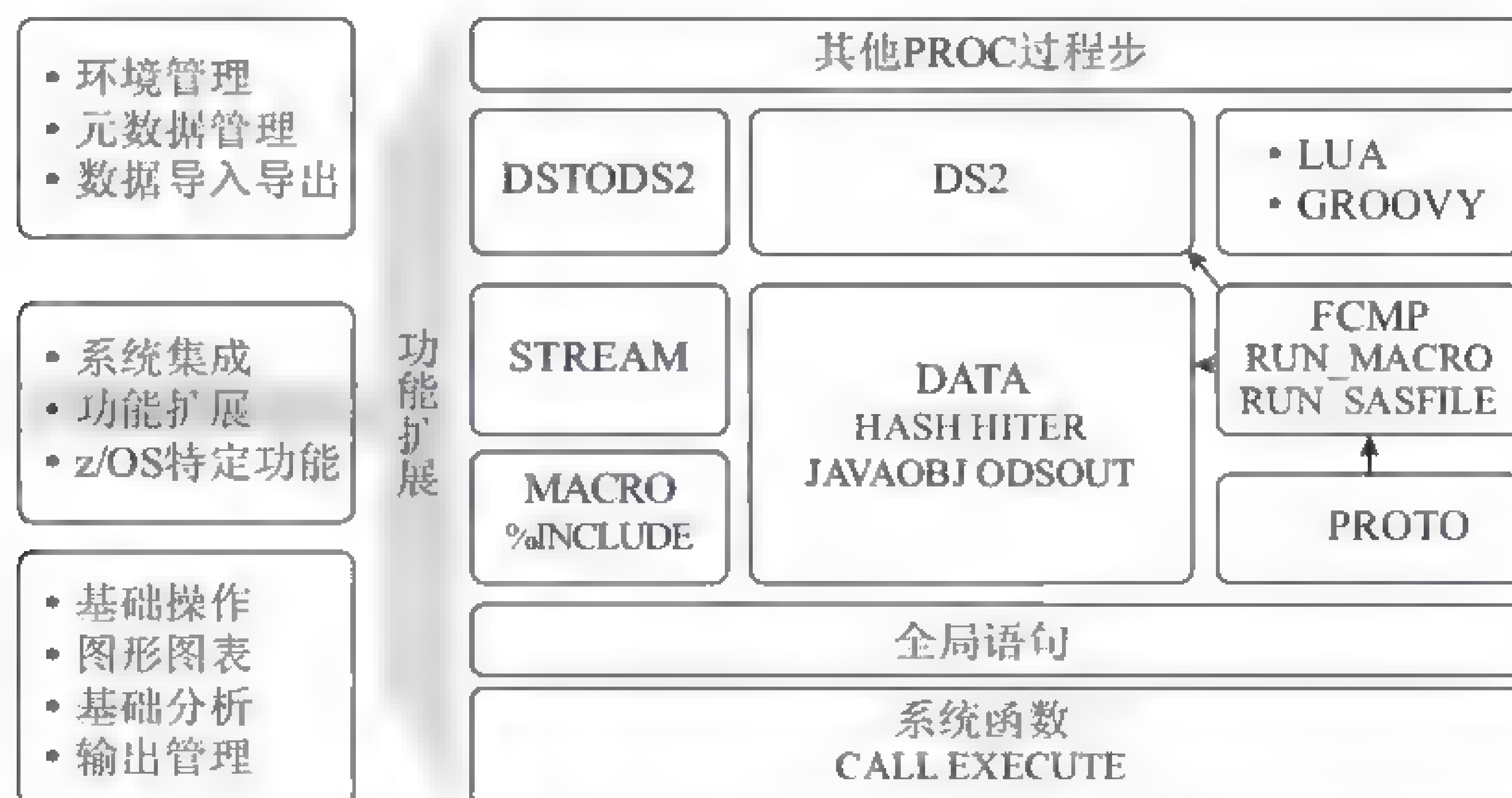


图 18-44　Base SAS 各模块之间的结构关系

1. 基础操作PROC

（1）CONTENTS 显示数据集的内容，并打印 SAS 逻辑库的目录。

（2）DATASETS 管理各种 SAS 文件的工具过程步，包括数据集的复制、重命名、修复、删除等，也包括追加数据集和索引，以及列出数据集属性口令等信息。

（3）APPEND 将观测从一个数据集追加到另一个数据集的末尾。

（4）COPY 将一个或多个数据集从一个 SAS 逻辑库拷贝到另一个 SAS 逻辑库；该过程步支持顺序数据访问类型的 SAS 逻辑库。

（5）DELETE 删除 SAS 逻辑库的成员，成员可以包括永久或临时 SAS 文件，以及加载到 CASLIB 中的 CAS 表。该过程步可批量删除数据集，或按成员类型进行删除等。

（6）COMPARE 比较两个数据集的内容，或对来自不同数据集，或者同一数据集的变量进行比较。

（7）CATALOG 管理 SAS 目录册的过程步，包括创建 Catalog，复制或选择条目入口，重命名、交换与删除条目，修改或删除条目入口的描述文本等。该过程步是一个交互式语句驱动的过程步。

（8）REGISTRY 管理 SAS 注册表的过程步，包括 SASHELP 逻辑库中的系统注册表和 SASUSER 逻辑库中的用户注册表。包括导入 / 导出，比较删除以及卸载注册表文件等注册表操作。

（9）FORMAT 创建用户自定义输入 / 输出格式。可将输入 / 输出格式存储为 SAS 数据集，或从 SAS 数据集中加载用户自定义的输入 / 输出格式。

（10）FMT2ITM 将一个或多个存储 SAS 格式信息的 SAS CATALOG 转换为 CAS 中可用的单个 ITEMSTORE 文件，它是将传统 SAS 格式信息导入到 CAS 服务器上进行重用的唯一方法。

（11）DATEKEYS 用于时间序列中需要用名字来引用日期时，处理单个日期或者一组日期变量，可用于标记日期键值有关的时间区间，也可标记假期和销售活动等。

（12）PWENCODE 对口令进行编码或者将文本数据进行混淆，该混淆是一个并非基于私钥的可逆操作。比如要在 SAS 代码中需要提供口令访问关系型数据库，SAS/CONNECT, SAS/SHARE, SAS IOM 或 SAS Metadata 服务器等对象时，可用此过程步把访问口令加密后放在 SAS 代码，可避免密码泄露和遗失。

（13）FSLIST 在 SAS 会话环境中浏览非 SAS 数据集格式的外部文件，可用于检查文件内容或者从 FSLIST 窗口中复制文本到 SAS 程序窗口中。

2. 数据导入导出PROC

（1）EXPORT 从 SAS 数据集中读入数据，导出到外部数据源。从 SAS 9.4 版本开始支持导出为空格 / 逗号 /TAB 分隔符的文件和 JMP 文件。

（2）IMPORT 从外部数据源导入数据生成 SAS 数据集。从 SAS 9.4 开始支持导入分隔符文件和 JMP 文件。

（3）CONVERT 将 BMDP, OSIRIS 系统文件或者 SPSS 导出文件转化为 SAS 数据集。

（4）CPORT 将 SAS 逻辑库，SAS 数据集以及 SAS Catalog 导出到顺序文件格式的迁移文件（Transport 文件）文件中。

（5）CIMPORT 导入由 CPORT 过程步导出的迁移文件（Transport 文件），迁移文件可包含 SAS 逻辑库、SAS 数据集以及 SAS Catalog 等 SAS 对象。

（6）CDISC 用 ODM 语句导入/导出符合 CDISC ODM 1.2 规范的 XML 文档，或者对一个符合 CDISC SDTM 3.1 规范的 SAS 数据集，根据 CDISCSDTM 领域定义执行数据内容校验。

（7）SQOOP 使用 Apache Sqoop 在 Hadoop 和关系型数据库管理系统（RDBMS）之间传输数据，也可在 SAS 会话中使用 SQOOP 过程步访问 Apache Sqoop 以在一个数据库和 HDFS 之间传输数据，从而实现从 SAS 应用中用 ApacheOozie Workflow Scheduler for Hadoop 提交 Sqoop 命令到 Hadoop 集群上。该过程步为 Sqoop 任务定义 Oozie 工作流，然后使用 RESTful API 提交到 Oozie 服务器。

（8）SCOREACCEL 为 DATA 步和 DS2 模型发布和评分提供 CAS 服务器接口。模型可以被发布到 CAS 或外部数据库上执行。也能从 CAS 中发布模型代码到一个外部数据库上后以 SAS Embedded Process (EP) 方式执行。它支持 Teradata 和 Hadoop 。发布 DATA 步模型代码时该过程步会自动翻译为 DS2 代码来执行。

（9）MIGRATE 将 SAS 逻辑库中的成员迁移到当前 SAS 版本中，包括数据集、数据视图、CATALOG、ITEMSTORE 和多维数据库 MDDB 等，但该过程步不支持编译后存储的 DATA 步和 SAS 宏，SAS 程序代码以及 SPD（Scalable Performance Data）引擎数据集。

3. 功能扩展PROC

（1）FCMP 函数编译器（Function Compiler）过程步用来创建，测试和存储 SAS 函数，CALL 例程和子例程，该函数可用于 DATA 步和诸多 PROC 步中。FCMP 采用的语法与 DATA 步稍有不同，编译输出为 SAS 数据集格式。FCMP 允许创建可重用的函数或例程进行复杂的读写操作。FCMP 过程步使用 SAS 语言编译器进行编译和执行，编译子系统生成机器代码运行。如指定代码生成优化选项 CMPOPT 可优化机器语言代码执行效率。FCMP 函数可用于 DATA 步，WHERE 语句，ODS 子系统以及这些过程步中：CALIS、OPTMODEL、FORMAT、PHREG、GA、QUANTREG、GENMOD、REPORT（仅COMPUTE块）、GLIMMIX、MCMC、SEVERITY、MODEL、SIMILARITY、NLIN、SQL（仅支持不带数组参数的函数）、NLMIXED、SURVEYPHREG、NLP、VARMAX、OPTLSO 和 SAS Risk Dimensions 过程步。

（2）PROTO 让用户能够以批处理模式注册 C/C ++ 编写的外部函数。这样就可以在 SAS 代码中调用 C/C++ 函数、数据结构和类型。一旦 C 语言函数在 PROC PROTO 中注册后，它们可以在 FCMP 过程步的函数或子例程中被调用，也可以在 COMPILE 过程步的函数、子例程以及方法块中被调用。

（3）DS2 使用户能在 Base SAS 会话中编写 DS2 代码，支持 SAS 9.4 和 SAS Viya。适用于高级数据操作，支持变量作用域，自定义方法，ANSI SQL 数

据类型和自定义包（对象）等。DS2 过程步的 SET 语句支持嵌入 FedSQL 语法，从而在运行时能在 DS2 和所支持的数据库间动态交换数据。DS2 可支持与 SAS 服务器，DBMS 数据源以及 CAS 服务器等类型。

（4）Lua 允许在 SAS 代码中运行 Lua 编程语言代码，甚至外部 Lua 脚本文件。在 Lua 语句中可调用大部分 SAS 函数和 FCMP 函数，也可以在 Lua 代码中提交 SAS 代码运行，Lua 中也可以调用 CAS 动作和读取 VARCHAR 数据。（注：Lua 嵌入式脚本语言可运行在任何有标准 C 编译器的平台上，包括所有的 UNIX、Windows、移动操作系统 Android、iOS 等。它具有语法简单、计算快速和自动内存管理等特性。）

（5）GROOVY 在 SAS 代码中利用 Java 虚拟机上执行 Goovy 动态语言代码，可将 Groovy 语句解析为 Groovy 类对象并运行，这些对象在其他 PROC GROOVY 语句或对 DATA 步中的 Java Obj 对象可见。

（6）STREAM 可处理包含 SAS 宏规范的任意文本输入流，其中的宏被展开和执行，而其他文本则保留不变。输出流可发送到一个外部文件或 SAS 输出目标。

（7）SCAPROC 是 SAS 代码分析器（Code Analyzer）过程步，它从一个 SAS 作业中捕获输入输出和宏符号等信息，将它写入到某个指定的文件中；也可生成在网格环境上可并行执行的作业。它也可以在命令行启动 SAS 作业时通过 -initstmt "proc scaproc; record 'myjob.txt'; run;" 启动 SAS 代码分析器。

（8）DSTODS2 将 DATA 步程序转换为 DS2 代码以使用 DS2 语言的更多优良特性，其输入输出为程序源代码，此过程不支持 z/OS 操作系统。

4. 系统集成PROC

（1）SQL 可在 SAS 代码中嵌入 SQL 语句操纵数据，它可连接到 DBMS 提交查询和非查询 SQL 语句，不支持 CAS 服务器。

（2）FEDSQL 允许在 Base SAS 会话中提交 FedSQL 语言代码到服务器上执行，它符合 ANSI SQL:1999 核心规范，支持扩展数据类型 DECIMAL/INTEGER/VARCHAR 以及其他 ANSI 1999 核心规范兼容的专属扩展。FedSQL 提供的数据访问技术支持伸缩性，多线程和高性能的访问方式，还可跨多个数据源访问，管理和共享关系型数据。当数据量很大时它默认支持多线程优化算法。FedSQL 是与厂商无关的 SQL 实现，兼容各种数据库和数据源厂商，它支持 SAS 服务器，DBMS 数据源和 CAS 服务器（从 9.4M5 开始）。最大的优点是单个 FedSQL 查询可从多个数据源读取数据，返回单个结果集。

（3）HTTP 发送 HTTP 协议请求，不但支持标准 HTTP 方法还支持任何 HTTP/1.1 标准的方法，包括持久化链接、Cookie 缓冲、EXPECT 100 CONTINUE 支持和身份验证类型等特性。它通过文件引用发送字符数据或者通过 HEADERS 语句发送名值对，甚至是外部格式化文件。该过程步让 SAS 是读取网络数据的众多方式之一。

（4）XSL 将 XML 文档从一种形式变换为另一形式，如 HTML，文本文件或其他 XML 文档类型。它读入一个 XML 文档，根据 XSL 风格表单（Extensible Stylesheet Language）进行 XML 数据变换，并输出到另一个文件。它使用 Saxson-EE 9.3 标准处理 XML 文件，XSLT 处理器则实现了 XSLT 2.0 标准。

（5）SOAP 用于处理 Web 服务的调用接口，它从一个文件读取 XML 输入，并将 XML 输出到另一个文件引用。一个服务请求相关的 SOAP 消息 XML 文档会被作为文件引用内容的一部分，它输入的 XML 可为 SOAPEnvelope 元素或 SOAPEnvelope 内调用 Web 服务的元素，是 SAS 中调用 Web 服务的接口。

（6）JSON 过程步是 JSON 文件格式处理接口，允许将 SAS 数据集中的数据输出到外部 JSON 文件，它可控制输出格式，也可以输出额外的数据到外部文件和控制 JSON 容器。

（7）HADOOP 通过执行 ApacheHadoop 代码跟 Hadoop 数据进行交互，Apache Hadoop 是 Java 开发的分布式海量数据存储和处理的开源框架。它可以和 Hadoop 内控制集群节点上的作业控制服务 Hadoop Job Tracker 打交道。HADOOP 过程步可从 SAS 语言中提交 HDFS 命令、MapReduce 程序以及 Pig 语言代码。

（8）INFOMAPS 用于程序化创建 Information Map，包括对现有的 Information Map 增加新的数据源、数据项、过滤器、文件夹和关系等，也可改变一个 Information Map 中现有除数据源以外的所有元素的定义。

（9）HDMD 为存储在 HDFS 文件系统上的文件生成描述其内容的 XML 元数据，这样 SAS/ACCESS HADOOP 接口和 HPA 过程步在不依赖于其他元数据储存库（如 Hive）的元数据就能直接读取 Hadoop 数据。

（10）S3 执行 Amazon S3 平台上的对象管理，比如创建 Buckets 并将文件加入，SAS 支持文件和目录对象类型。它需要 AWS Key ID 或安全令牌。当数据大于 5 MB 时 SAS 会启动额外的线程以提供更快的并行传输速度。

5. 图形图表PROC

（1）CHART 生成简单的垂直和水平条形图、饼图、星图，可用于数值或字符型。需要控制字体颜色时应该使用 SAS/GRAPH 产品的 GCHART 生成各种颜色和字体的图表。

（2）PLOT 将每个观测的两个变量进行简单投点（如 plot x *y; ）生成简单的图形图表。

（3）TIMEPLOT 在时间区段上投影一个或多个变量（垂直轴为时间，水平轴为观测值）形成简单的与时间有关的投点图。

（4）CALENDAR 以日历格式显示数据集中的数据，可生成计划日历或汇总日历。包含一些项目管理调度工具有关的特性。

（5）EXPLODE 为文本输出文类似于 ASC II 艺术的字符矩阵，提供空格、字符密度以及下划线控制等选项。

（6）SGPLOT 创建一个或多个投点图并将它们堆叠在相同的坐标轴上，包括直方图、回归投点图等。它具有极其丰富的语句可以绘制各式各样的图形。

（7）SGSCATTER 创建多面板的散点图，也支持在散点图上对点进行拟合，以及绘制椭圆曲线等。

（8）SGMAP 地图处理过程步，它融合了 SAS/GRAPH 过程步 GPROJECT, BASE 过程步 GEOCODE 和 MAPIMPORT 等过程步的数据操作能力和 ODS Graphics 的功能，它可创建地图并在上面叠加文本，散点及泡泡图等。

（9）SGPANEL 可根据一个或多个分类变量创建图形单元面板，比如绘制一组网格状排列的投点图。它是对图形进行面板控制的主要过程步。

（10）SGDESIGN 基于一个或多个输入数据集以及一个用户自定义 ODS 图形设计器文件（SGD 文件）生成图形输出，SGD 文件由 SAS 的 ODS 图形设计器应用程序创建。如果 SGD 文件中定义了动态变量，用户可在 SGDESIGN 过程步中用 DYNAMIC 语句实现基于动态数据的图形输出，而不是 SGD 文件中指定的一个或多个数据集。

（11）SGRENDER 基于 GTL 模板语言创建的模板绘制图表，通常称为 StatGraph。GTL 是 ODS 图形过程步之外制作任何复杂统计图和创建定制化布局的强大工具。它也可以基于 SAS ODS Graphics Editor 工具输出的文件绘制图形图表。

6. 基础分析PROC

（1）SORT 根据一个或多个变量对数据进行排序，输出一个数据集。

（2）RANK 基于计算一个或多个数值变量对记录进行排名，且输出计算结果到另一个数据集。

（3）TRANSPOSE 转置数据，将数据集中的行和列（观测和变量）进行旋转。

（4）STANDARD 数据标准化过程步，将 SAS 数据集中的变量标标准化为给定均值和标准差的数据并输出。

（5）MEANS 汇总数据并对所有观测数据或分组数据计算描述性统计量，计算包括基于矩的描述性统计量、估计分位数（包括中值）、均值的置信区间、识别极端值、执行 t 检验等。该过程步可以显示输出，也可以使用 OUTPUT 语句输出统计量到数据集中。与 SUMMARY 非常相似。

（6）SUMMARY 对数据集中的所有观测或分组观测计算各描述性统计量，它是一个数据汇总工具过程步且支持 CAS 环境。

（7）FREQ 为单个变量或多个变量生成频数表和交叉表。对 2- 向表计算检验和关联程度的度量。对于 n- 向表它计算层内和层间的统计量来进行分层分析。

（8）TABULATE 以表格形式显示描述性统计量，支持简单的显示到高度定制化

的表型报告。在对变量的数值进行分类和建立层次关系方面具有很好的灵活性，适用于标签和格式化变量，以及各种统计量。它输出许多与 MEANS、FREQ、REPORT 相同的描述性统计量。

（9）UNIVARIATE 单变量分析过程步，可生成各种统计量和图表：基于矩的描述性统计量（包括偏度和峰度），分位数和百分位数，频数表和极端值等；也可输出直方图（支持拟合，概率密度曲线和核密度估计）、累积分布函数图（CDF）和各种概率分布曲线、概率图、Q-Q 图、P-P 图，可用于比较数据分布和各种理论分布的差异。同时，它还可输出各种分布（包括正态）的拟合优度测试，在图表上嵌入汇总统计量等。它还支持对包含频数变量的数据集进行分析，可将汇总统计量，直方图区间和拟合曲线参数等数值输出到一个数据集。

（10）CORR 计算皮尔逊相关系数、三种非参数关联度量、多相关系数以及与这些统计量相关的概率。

7. 输出管理PROC

（1）PRINT 选择全部或部分变量，打印 SAS 数据集或 CAS 表中的观测。它既可以生成非常简单的列表，也可以创建高度定制化，对数值变量数据进行分组小计和总计的复杂报表。

（2）PRINTTO 用来为 PROC 步和 SAS 日志定义默认 ODS 目标之外输出目标。默认 SAS 过程步输出和 SAS 日志被定向到默认的 PROC 输出文件和 SAS 日志文件中，但 PRINTTO 可以将它们输出到一个外部文件或 SAS Catalog 条目（此时必须打开 ODS LISTING），利用它甚至可以在同一个 SAS 进程中将前面的 SAS 输出当作后续程序的输入数据进行处理。在 Windows 环境中，默认的 PROC 输出和 SAS 日志输出目标为“结果查看窗口”和“LOG 窗口”，但在交互线模式中，它们分别是作为报表输入“显示器”和显示每个步骤执行的“显示器”；而在非交互模式或批处理模式中，它们取决于主机操作系统和操作环境。有时为了避免生成大量的日志文件，可用此过程步来自定义日志文件格式。

（3）REPORT 它合并了 PRINT、MEANS、TABULATE 过程步和 DATA 步众多模块的特性为一体，用来生成一系列报告，可包括明细表和汇总表等功能。

（4）FORMS 生成信封标签、邮件标签、外部磁带标签、文件卡和其他任何有固定规则的打印表单，数据集中的每一个观测被打印到一个矩形的表格单元，即套打打印。

（5）DOCUMENT 过程步与 ODS DOCUMENT 语句配合使用可以将报告的单个元素（ODS 对象）进行存储、修改并重放。本过程步可对 PROC 步或单纯数据库查询的原始报表结果进行重排、复制或删除，而不需要重新执行分析过程步或者数据库查询。

（6）ODSLIST 用于创建列表类模板。包括可定制的文本模板列表，可定制内容的风格属性和格式化信息。可通过它的 DATA 选项绑定数据集与模板（不需要使用 DATA 步程序）。它主要用于生成 Power Point 和 e-Books 格式的 ODS 目标的内容。

（7）ODSTABLE 为 ODS 系统创建自定义的表格型输出模板，可用于除了 PRINT、REPORT 和 TABULATE 过程步之外的所有过程步。它比 PROC TEMPLATE 过程步 DEFINE TABLE 语句生成表格模板的方法更加简单。

（8）ODSTEXT 用于生成文本块模板，这些模板使用风格属性和 Format 来定制内容，用于创建列表和段落。

（9）TEMPLATE 是 Base SAS 中最复杂的过程步之一，主要包括如下功能。

①定制表模板：用于定制 SAS 输出表的外观，包括表头、表体、表尾和列的模板风格。除了 PRINT、REPORT、TABULATE 之外的所有 PROC 的 SAS 输出表格风格受该表格模板的控制。

②定制交叉表模板 (DEFINE CROSSTABS)：用于定制 FREQ 输出的交叉表外观，默认交叉表使用的是 CrossTabFreqs 模板进行渲染，可通过本过程步创建定制的表模板来改变外观。

③创建 ODS Graphics 图 (DEFINE STATGRAPH)：SAS 9.2 引入 GTL 图形模板语言来定义清晰有效的统计图形图表，GTL 支持生成各种各样的投点图，模型拟合图、分布图、比较图、预测图等。默认输出 24 位 PNG 文件，支持抗锯齿和透明背景。其核心是使用一个灵活的模板 STATGRAPH 来灵活构建图表， STATGRAPH 也是用 TEMPLATE 过程步来定义的。

④创建 Style 模板（DEFINE STYLE）：定制 SAS 输出的外观，它可创建和修改风格模板，而 ODS 使用这些定制模板对输出图表和报告进行格式化。

⑤创建标记语言标签集：标签集（Tagsets）是控制 SAS 输出形成标记语言文件的模板，它可为 ODS 指定标记语言的输出格式（如 XML、HTML、XSL 等输出），用户也可创建自定义的标记语言标签集（Markup Language Tagsets）。

⑥管理模板存储：SAS 创建的模板项存储在系统逻辑库 SASHELP.Tmplmst 中，用户也可以将自定义模板存储在系统中任何可写的特定逻辑库中。

8. 环境管理PROC

（1）OPTIONS 列出当前 SAS 的系统选项，输出到 SAS 日志中。SAS 系统选项控制 SAS 输出、文件处理、数据集处理、操作系统的交互以及其他全局系统行为。

（2）OPTSAVE 将 SAS 系统选项设置保存到 SAS 注册表或者单个 SAS 数据集中，结合 OPTLOAD 过程步可以实现将别的 SAS 运行环境设置完全恢复到当前 SAS 会话中。

（3）OPTLOAD 从存储在 SAS 注册表或一个使用 OPTSAVE 过程步生成的 SAS 数据集中读取 SAS 系统选项设置，并立即生效。

（4）PRESENV 将所有全局语句和宏变量保存到外部文件中，临时数据集 WORK 中的数据和 SAS 宏 CATALOG 被写入辅助目录。这样就可以实现 SAS 运行的断点继续功能。重启的 SAS 会话可以通过恢复保存的全局语句和 SAS 宏变量以及临时数据集，完全恢复上次运行的系统状态。

（5）LOCALEDATA 定制化系统 Locale 数据，包括查看、打开、修改和存储自定义 Locale 数据。

（6）DBCSTAB 生成 SAS 支持的双字节字符集（DBCS）转换表，用于当前正在使用的 DBCS 编码系统具有非标准的码点表，或者是 SAS 不支持的 DBCS 编码系统。

（7）TRANTAB 创建，编辑和显示定制化翻译表，其数据存在 SASUSER.Profile 目录册中，也可修改 SAS 系统翻译表（在 SASHELP.Host 目录册中）。翻译表用于不同编码系统之间进行正向 / 逆向码点值转换。

（8）FONTREG 将操作系统中的 FreeType 和各种类型的字体注册到 SAS 注册表中，这样这些字体就可以在 SAS 输出中使用。

（9）PRTDEF 在批处理模式中为当前站点上的单个用户或所有用户在 SAS 注册表中定义打印机。使用 USESASHELP 选项创建偏好的打印机定义对所有 SAS 用户可用，否则仅限当前用户可用。

（10）PRTEXP 为复制和修改目的，从 SAS 注册表中解析打印机属性信息，可写入 SAS 日志或 SAS 数据集中，也可从 SASHELP 或整个 SAS 注册表中查找这些属性。

（11）QDEVICE 生成关于图形设备和通用打印机有关的各种报告，包括颜色支持，默认输出大小、边距大小，分辨率，支持的字体，硬件符号，硬件填充类型，硬件线条样式，设备选项等。用于在特定应用中选择所需的图形或打印机硬件设备，结果可输出到 SAS 日志或者 SAS 数据集。

（12）JAVAINFO 显示 SAS 运行系统中 Java 环境有关信息，用于诊断 SAS Java 环境的配置，验证 SAS 系统中的 Java 环境是否正常工作。

（13）PMENU 定义用户交互菜单，这些菜单可用于 DATA 步窗口，宏窗口，SAS/AF 和 SAS/FSP 窗口，或任何其他可定制菜单的 SAS 应用。菜单可在命令行窗口中运行 PMENU 命令来激活。

（14）DISPLAY 执行 SAS/AF 应用，它执行存储在 SAS Catalog 中并使用 SAS/AF 产品的 BUILD 过程步编译的 SAS/AF 应用。

9. 元数据管理PROC

（1）METADATA 通过发送 XML 文本到 SAS Metadata Server 进行元数据读写操作，也可通过 SAS 开放元数据接口（Open Metadata Interface， OMI）服

务器状态监视元数据服务器及其配置信息。

（2）METALIB 为 SAS 数据集（或视图）和 DBMS 数据表生成元数据信息，在 SAS Metadata Server 上创建或者更新数据表、数据列、索引以及一致性约束检查。

（3）METAOPERATE 在批处理模式中执行管理 SAS Metadata Server 有关的任务，包括对 SAS 元数据储存库的删除、清空、注销、刷新以及临时暂停或停止 SAS 元数据服务器。

（4）AUTHLIB 管理元数据绑定逻辑库的工具过程步，包括创建、修改、净化、修复、删除和报告等功能。

10. z/OS 大型机特定PROC

（1）ITEMS 创建并读写 SAS ITEMSTORE 格式的文件，本过程步仅限 IBM 大型主机 z/OS 平台上。

（2）PDS 列出，删除或重命名已分区的数据集的成员，包括两种类型的分区数据集：一种是源代码，SAS 宏，存储在 Catalog 中的过程步和其他数据；另一种是仅包含 Load 模块的 Load 逻辑库。本过程步仅限 IBM 大型主机 z/OS 平台上。

（3）PDSCOPY 将分区数据集从磁盘，磁带之间进行复制，本过程步仅限 IBM 大型主机 z/OS 平台上。

（4）SOURCE 用于 z/OS 平台上读取 PDS 或 PDS Library 并生成顺序输出，用于备份和处理源逻辑库数据集。

（5）RELEASE 在一个磁盘数据集的末尾释放未用空间，可用于大多数顺序或分区数据集，而不仅是包含 SAS 数据集的 SAS 逻辑库。本过程步仅限 IBM 大型主机 z/OS 平台上。

（6）TAPECOPY 将整个磁带卷，或文件从一个或多个磁带卷复制到一个输出的磁带卷。本过程步仅限 IBM 大型主机 z/OS 平台上。

（7）TAPELABEL 将一个 IBM 标准的标记磁带卷的标签信息写出到 SAS 过程步输出文件，包括数据集名称、DCB 信息、数据集历史等。

18.5.3 SAS/STAT过程步速查

表 18-4 为笔者精心整理的全部 SAS/STAT 过程步分类表：纵向为按照字母顺序排序的 101 个 PROC 步（包括已经出现在 Base SAS 产品中的 PROC FREQ），横向为对应 PROC 的主要数据分析领域。读者可从此表迅速查找特定分析领域对应的 SAS 过程步进行数据分析。

表 18-4　数据分析方法与 SAS/STAT 过程步对应关系（A-L，共 51 条）

PROC	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
ACECLUS																											Y														1
ADAPTIVEREG																				Y						Y			Y													3
ANOVA										Y																														1
BCHOICE							Y																													Y				2
BOXPLOT			Y																																					1
CALIS													Y																					Y							2
CANCORR														Y		Y																								2
CANDISC																		Y																						1
CATMOD										Y	Y																												2
CAUSALMED																																	Y								1
CAUSALTRT																																	Y								1
CLUSTER																											Y														1
CORRESP														Y		Y																					Y		Y		4
DISCRIM																		Y																						1
DISTANCE																											Y														1
FACTOR														Y	Y																								Y		3
FASTCLUS																											Y														1
FMM							Y				Y																				Y										3
FREQ				Y								Y														Y																3
GAM																										Y																	1
GAMPL																										Y																	1
GEE													Y																	Y												2
GENMOD				Y			Y				Y																		Y												4
GLIMMIX																														Y	Y										2
GLM										Y								Y																						2
GLMMOD																				Y		Y																				2
GLMPOWER								Y																																	1
GLMSELECT																												Y	Y												2
HPCANDISC																		Y																						Y		2
HPFMM																																	Y								Y		2
HPGENSELECT																												Y	Y												Y		3
HPLMIXED																															Y										Y		2
HPLOGISTIC																												Y														Y		2
HPMIXED																															Y										Y		2
HPNLMOD																																								Y		1
HPPLS																																								Y		1
HPPRINCOMP																																								Y		1
HPQUANTSELECT																													Y													Y		2
HPREG																				Y									Y														Y		3
HPSPLIT																																								Y		1
ICLIFETEST																									Y																		1
ICPHREG																				Y					Y																		2
INBREED										Y																														1
IRT																																							Y		1
KDE		Y																						Y																		2
KRIGE2D																																						Y			1
LATTICE										Y																														1
LIFEREG							Y												Y					Y																		3
LIFETEST																									Y																		1
LOESS																				Y						Y																	2
LOGISTIC				Y							Y								Y		Y																				4

表 18-4 数据分析方法与 SAS/STAT 过程步对应关系（M -Z，共 50 条）

（续）

PROC名称	据标准化	[illegible]	述性统计	确推断	样调查与分析	成组贯序设计与分析	贝叶斯分析	功力和样本	[illegible]	方差分析	类别数据分[illegible]	缺失值数据分析	元分析	因子分析	[illegible]	[illegible]	判别分析	归分析	[illegible]	辑回归	稳健回归	分位数回归	[illegible]	[illegible]	非[illegible]分[illegible]	聚类分析	[illegible]型	模[illegible]选择	纵[illegible]数据分[illegible]	混[illegible]模型	有[illegible]混合模型	结[illegible]方程模型	[illegible]析	[illegible]分匹配	后[illegible]	[illegible]研究	[illegible]析	心[illegible]分析	高[illegible]版本	汇总
MCMC							Y					Y																													2
MDS													Y																							Y		Y		3
MI												Y																												1
MIANALYZE												Y																												1
MIXED																													Y											1
MODECLUS																										Y														1
MULTTEST				Y																																				1
NESTED										Y																														1
NLIN																		Y	Y																					2
NLMIXED																														Y										1
NPAR1WAY				Y						Y															Y															3
ORTHOREG																		Y																						1
PHREG							Y											Y					Y							Y						Y				5
PLAN										Y																														1
PLM																		Y																	Y					2
PLS																		Y																						1
POWER								Y																																1
PRINCOMP													Y		Y																							Y		3
PRINQUAL													Y		Y																					Y				3
PROBIT											Y																													1
PSMATCH																																		Y						1
QUANTLIFE																					Y		Y																	2
QUANTREG																		Y			Y																			2
QUANTSELECT																												Y												1
REG																		Y																						1
ROBUSTREG																		Y			Y																			2
RSREG																		Y																						1
SCORE																																			Y					1
SEQDESIGN						Y																	Y																	2
SEQTEST						Y																	Y																	2
SIM2D																																					Y			1
SIMNORMAL									Y																															1
SPP																																					Y			1
STDIZE	Y																																							1
STDRATE			Y																																					1
STEPDISC																	Y																							1
SURVEYFREQ					Y																																			1
SURVEYIMPUTE					Y							Y																												2
SURVEYLOGISTIC					Y															Y																				2
SURVEYMEANS					Y																																			1
SURVEYPHREG					Y													Y					Y																	3
SURVEYREG					Y													Y																						2
SURVEYSELECT					Y																																			1
TPSPLINE																		Y						Y																2
TRANSREG																		Y	Y								Y									Y		Y		5
TREE																										Y														1
TTEST										Y																														1
VARCLUS																										Y														1
VARCOMP																														Y										1
VARIOGRAM																																					Y			1
汇总	1	1	2	5	7	2	6	2	1	9	6	6	6	1	2	2	4	21	2	3	3	0	9	5	3	7	6	4	4	6	2	1	2	1	2	6	4	6	12	101

SAS/STAT 全部过程步的简要说明如下文所述。为了跟上面的表格配合，所有的 PROC 步是按照字母顺序排列的（其中 FREQ 过程步在 Base SAS 产品的 PROC 列表中出现过）。

（1）ACECLUS 基于近似协方差估计 (ACE) 进行聚类。它假定聚类的簇是具有相等协方差矩阵的多元正态，ACECLUS 能够获得簇内协方差矩阵的近似估计而不需要知道簇的数量和簇成员的数量。在进行 FASTCLUS 和 CLUSTER 过程步处理之前用于预处理数据很有用。

（2）ADAPTIVEREG 根据 FriedMan (1991) 定义的方法拟合多元自适应回归样条曲线，它是一种结合回归样条和模型选择方法的非参数回归技术。它没有假设参数化模型也不要求指定结值来构建回归样条，它自适应地为不同变量选择恰当

结值，并通过模型选择技术来简化模型，从而自适应地构造样条基函数。

（3）ANOVA 对来自各种试验设计的平衡数据进行方差分析（ANOVA)。其中连续的响应变量（因变量）在由分类变量（独立变量）表示的试验条件下进行测量。假设响应的变化是由于分类中因素的效应，随机误差考虑了剩余的变化。

（4）BCHOICE 贝叶斯选择过程步，它对离散选择模型进行贝叶斯分析。离散选择模型用于市场研究中模拟在替代产品和服务中进行选择的决策者进行建模。决策者可以是人，家庭、公司等，替代品可以是产品，服务和行动，或任何其他选项，或者其他必须做出选择的项。为决策提供备选方案的集合称为选择集。

（5）BOXPLOT 绘制由一系列并排的盒形图和线条组成的箱线图，可揭示一组数据的若干主要统计量，包括平均值、四分位数、最小值和最大值。

（6）CALIS 用于社会和行为科学中的重要统计工具——结构方程模型，它表达了一个所有变量都是随机变量的变量系统之间的关系，变量可以是显性的观测变量，也可以是不能观测所得的隐藏变量，它们服从近似多变量的正态分布。CALIS 采用最大似然（ML) 估计和广义最小二乘（GLS) 估计。

（7）CANCORR 典型相关分析，部分典型相关分析和典型冗余分析。用于分析两个变量集合之间关系的多重相关性的泛化。在多重相关中，关系通过一个由解释变量组的线性组合与单个因变量来揭示他们之间的关系，而典型相关中是一个变量集合的显现组合与另一个变量集的线性组合之间的关系，这些线性组合成为典型变量。两组变量在统计模型中是对称的，任何一组都可以被当作解释变量或响应变量。简单相关和多重相关可以被看作是典型相关在一个或两个集合都只包含一个变量时的特例。

（8）CANDISC 典型判别分析，是主成分分析和典型相关分析有关的降维技术。典型判别分析寻找在类或组之间能提供最大区分的定量变量的线性组合，它生成量化变量的线性组合称为典型变量，汇总类间变化的方式与主成分汇总总变异的方式相同。

（9）CATMOD 对以列联表表示的类别数据进行分类数据建模，它将线性模型拟合为响应频数的函数，可用于线性建模，对数线性建模，逻辑回归，重复测量分析等。对一般线性模型中使用加权最小二乘法（WLS) 估计，对数线性模型和广义 Logits 分析使用最大似然法估计。

（10）CAUSALMED 对观测到的数据中的因果关系中介效应进行分析和估计。它涉及处理变量 T（医学分析中称为暴露），中介变量 M 和输出变量 Y，以及一组混杂了 TMY 之间关系的类别或连续型的预处理或背景协变量。

（11）CAUSALTRT 在二元处理 T 中对连续或离散结果 Y，估计处理的平均因果效应。它可以估计两种因果效应：平均处理效果 ATE 和已处理的平均处理效果 ATT。

（12）CLUSTER 对数据集中的观测大多用 11 种聚类方法进行层次型聚类，数据可以是坐标或距离，默认使用欧氏距离。用户也可用 DISTANCE 过程步计算非欧几里得距离供本过程步做进一步分析。

（13）CORRESP 执行简单对应分析和多重对应分析。对应分析可以用查找交叉表或列联表的行和列的低维图像表示，每一行和列对应的单元中的频数，在图中由一个点进行表示。该过程步的输入数据可以是原始二分类或多分类的原始分类响应数据，或双向列联表。

（14）DISCRIM 判别分析，对一个或多个定量变量，以及一个定义观测分组的分类变量的数据集中所有观测，建立判别函数（判别标准）来将观测分成一个或多个分组。判别标准可用于其他数据集进行类别判定和预测。

（15）DISTANCE 对数据集中的观测之间计算各种距离和相似系数，变量可包含数值型或者字符型变量。

（16）FACTOR 因子分析和主分成分析过程步，支持旋转。输入数据可以是多元数据、相关矩阵、协方差矩阵，以及因子模式或者评分系数矩阵。可基于相关性矩阵或者协方差矩阵且可将大部分结果输出到 SAS 数据集。

（17）FASTCLUS 对一个或多个定量变量，基于距离进行不相交的聚类分析，每个观测只能属于有且仅有一个组。它不形成树形结构而是平板分组。如果要对不同数量的簇进行单独分析可独立运行它，或者用于对样本容量较大的数据集进行分层聚类。它可用于查找初始簇作为于层次聚类 CLUSTER 过程步的输入。

（18）FMM 有限混合模型，每个响应变量来自若干个未知概率的分布，它将统计模型拟合为服从有限分布的混合体。传统统计分布可看作是 FMM 在有且仅有一个分量时的特例。

（19）FREQ 形成单向或 N 向的频数表或列联表（交叉表），对双向表计算关联的度量和检验，对 N 向表它计算层内和层间的统计量来进行分层分析。

（20）GAM 拟合广义加性模型。

（21）GAMPL 基于低阶回归样条，拟合广义加性模型 (GAM)。

（22）GEE 提供加权广义估计方程。

（23）GENMOD 拟合广义线性模型。

（24）GLIMMIX 拟合广义线性混合模型。

（25）GLM 拟合一般线性模型，也用于方差分析的线性模型。

（26）GLMMOD 构造一般线性模型的设计矩阵；它本质上为 GLM 过程步构建建模前端。

（27）GLMPOWER 对线性模型进行前瞻性分析和样本容量分析。

（28）GLMSELECT 在一般线性模型框架下进行效应选择。

（29）HPCANDISC 典型判别分析的高性能版本。

（30）HPFMM 有限混合模型的高性能版本，对有限混合响应分布或者单变量分布数据拟合统计模型。

（31）HPGENSELECT 为广义线性模型提供模型拟合和模型构建。它适于指数类型的标准分布模型，如正态分布、泊松分布和 Tweedie 分布，是 GENSELECT

过程步的高性能版本。

（32）HPLMIXED 拟合多种混合线性模型，使拟合的模型可以对数据进行统计推断。

（33）HPLOGISTIC 对二元、二项和多项数据拟合逻辑回归模型，LOGISTIC 的高性能版本。

（34）HPMIXED 通过稀疏矩阵技术，拟合具有简单协方差分量结构的线性混合模型。

（35）HPNLMOD 用非线性最小二乘法或最大似然法拟合，为非线性回归模型的高性能版本。

（36）HPPLS 偏最小二乘法（PLS）拟合模型进行线性预测，为偏最小二乘法 PLS 的高性能版本。

（37）HPPRINCOMP 主成分分析高性能版本。

（38）HPQUANTSELECT 对分位数进行回归分析，拟合和执行效应选择。是分位数回归框架下的效应选择过程步 QUANTSELECT 的高性能版本。

（39）HPREG 对普通线性最小二乘模型，进行拟合和执行模型选择。

（40）HPSPLIT 构建基于树的统计模型，以进行分类和回归。

（41）ICLIFETEST 执行区间删失数据（Interval-Censored data）的非参数化生存分析。

（42）ICPHREG 用于将比例风险回归模型（proportional hazards regression models）拟合区间删失数据。也可用于故障时的无删失数据、右删失数据和左删失数据；总体成员的幸存时间被假定为服从风险函数 $\lambda_i(t)$。

（43）INBREED 用于计算系谱的协方差或近交系数（Inbreeding coefficients）。

（44）IRT 拟合项目反应理论（IRT）模型。

（45）KDE 执行单变量和双变量核密度估计。

（46）KRIGE2D 在两个维度上，执行普通克里金法（Ordinary Kriging）。它可处理各向异性和嵌套的 8 种变异函数模型：高斯、指数、球形、动力、立方、五球、正弦孔效应以及 Matérn 模型。

（47）LATTICE 对点阵设计的试验数据进行方差分析和简单协方差分析。

（48）LIFEREG 生存分析过程步，对无删失、右删失、左删失和区间删失的故障时间数据，拟合参数化模型。

（49）LIFETEST 生存分析过程步，通过乘积极限法（也称 Kaplan-Meier 法）或寿命表法（也称精算方法）计算幸存者函数的非参数估计，用于非参数化生存分析。

（50）LOESS 实现由 Cleveland, Devlin 和 Grosse (1988) 年开创的对回归曲面进行估计的非参数方法，由于不需要对回归面进行假设参数形式而具有很大的灵活性。适用于数据中包含异常值且需要一个稳健拟合方法时。

（51）LOGISTIC 逻辑回归过程步，拟合具有二元、定序或者定类因变量的回归模型。

（52）MCMC 执行通用的马尔可夫链蒙特卡罗（Markov Chain Monte Carlo）模拟拟合贝叶斯模型。贝叶斯统计与传统基于最频繁或经典方法不同的统计方法。

（53）MDS 拟合双向和三向、度量和非度量的多维标度模型（Multidimensional scaling），是对特定维度空间中一系列对象的坐标进行估计的系列方法。

（54）MI 对缺失值（Missing Value）进行多重插补。

（55）MIANALYZE 结合多个估计（Multiple Imputation）的分析结果，生成有效的统计推断，适用于分析数据集中包含缺失值时。

（56）MIXED 拟合具有固定和随机效应的一般线性模型，用于对数据做出统计推断，也用于方差分析中的混合模型。混合线性模型（Mixed Linear Model）为数据的均值，方差和协方差提供了灵活的建模方法。

（57）MODECLUS 基于非参数密度估计的几种算法之一，对 SAS 数据集中的观测进行聚类。

（58）MULTTEST 在对同一数据集执行多个假设检验时，从一组假设检验中通过调整 P- 值来解决多重检验问题。

（59）NESTED 随机效应方差分析，适用于具有嵌套（层次）结构和分类效果的试验数据。

（60）NLIN 非线性拟合回归模型，并采用非线性最小二乘法或加权非线性最小二乘法估计参数。

（61）NLMIXED 拟合非线性混合模型，适用于固定和随机效应都是非线性场景。

（62）NPAR1WAY 非参数单因子方差分析过程步，对单向分类中的位置和尺度差异进行非参数检验。

（63）ORTHOREG 回归正交设计，用最小二乘法拟合一般线性模型，针对病态数据，提供比其他过程步更精确的估计。

（64）PHREG 基于 Cox 比例风险模型（Proportional Hazards Model），对生存数据进行回归分析，常归于生存分析。

（65）PLAN 为因子试验进行试验设计和随机化计划，特别是嵌套和交叉试验随机块设计。

（66）PLM 对 SAS STAT 其他过程步生成的 ITEMSTORE 格式的数据作后拟合统计分析。

（67）PLS 偏最小二乘法（PLS）对模型进行拟合，过程步也使用任意线性预测方法拟合模型。

（68）POWER 对各种统计分析进行前瞻性分析和样本量分析。

（69）PRINCOMP 执行主成分分析过程步，输入数据可以是原始数据、相关矩阵、协方差矩阵或 SSCP 矩阵（Sum-of-Squares-and-crossproducts, SSCP）。

（70）PRINQUAL 执行定性数据、定量数据或它们的混合数据进行主成分分析（PCA）。

（71）PROBIT 对生物试验的量子响应数据或其他离散事件数据，计算回归参数的最大似然估值和自然（或阈值）响应率，包括 Probit、Logit、普通 Logistic、GoMpit 回归模型。

（72）PSMATCH 倾向性得分配对分析，提供各种各样的倾向性配对分析的工具。

（73）QUANTLIFE 生存分析过程步，对生存数据进行分位数回归分析。

（74）QUANTREG 拟合分位数回归模型，采用分位数回归，在对一个响应变量的条件分位数上，为协变量的效应建行建模。

（75）QUANTSELECT 实现分位数回归框架中的效应选择。

（76）REG 回归分析过程步，是回归分析中最通用，采用普通最小二乘法进行回归的过程步。

（77）ROBUSTREG 稳健回归，用于侦测异常值和在异常值的存在的情况下通过限定异常值作用来提供平稳结果。

（78）RSREG 使用最小二乘法拟合二次响应面回归模型（Response Surface Model）。响应面模型是一种一般线性模型，其重点关注拟合响应函数的特性，特别是最佳估计的响应值发生的地方。

（79）SCORE 将两个 SAS 数据集的值相乘，其中一个包含系数，另一个包含使用该数据集中的系数进行评分的原始数据。

（80）SEQDESIGN 用于临床试验的中期设计，临床试验是以人为目标检验新药或新处方的效应和安全性的试验。

（81）SEQTEST 对分组序列的临床试验进行中期分析。

（82）SIM2D 利用 LU 分解技术，对具有二维均值和协方差结构的高斯随机域进行空间模拟。

（83）SIMNORMAL 对一组正态相关或高斯随机变量进行条件和无条件模拟。

（84）SPP 对二维空间点的模式进行分析。

（85）STDIZE 专业的数据标准化过程步，通过减去一个位置度量（如均值）并除以一个尺度值（如标准差），对 SAS 数据库中的单个或多个数值型变量进行数据标准化。

（86）STDRATE 对研究总体直接计算标准化的比率和风险，用于流行病学研究和分析。

（87）STEPDISC 逐步判别分析过程步，对给定一个分类变量和几个定量变量进行逐步判别分析，以选择用于区分类之间的定量变量的子集。

（88）SURVEYFREQ 抽样调查与分析过程步，从具有分层、聚类和不等权重的复杂多层抽样设计中，产生 1 到 N 向频率表或交叉表。

（89）SURVEYIMPUTE 对数据集中的缺失值，用同一项中的观测值进行替换。

（90）SURVEYLOGISTIC 抽样调查与分析过程步，适用于通过极大似然法，对离散响应抽样数据进行逻辑回归的模型。模型可包括二元、定序或定类因变量。

（91）SURVEYMEANS 抽样调查与分析过程步，通过抽样调查计算统计量，来估计总体的特征。例如从分层、聚类和不等权的复杂多层抽样中，计算均值、总和、比例、分位数和比率等。

（92）SURVEYPHREG 抽样调查与分析过程步，是基于 Cox 比例风险模型的复杂抽样样本设计，对生存数据执行回归分析。

（93）SURVEYREG 抽样调查与分析过程步，对复杂抽样样本进行线性回归分析。

（94）SURVEYSELECT 抽样调查与分析过程步，提供多种方法来选择基于概率的随机样本。本过程步可以选择简单的随机样本，也可以根据复杂的多阶段设计进行抽样，包括分层、聚类和不等概率选择。当使用概率抽样时，调查总体中的每个单元都有一个已知被抽中的概率。

（95）TPSPLINE 使用惩罚最小二乘法来拟合非参数回归模型。它计算薄板平滑样条（Thin-plate smoothingsplines），来近似估计观察到的包含噪点的平滑多变量函数。本过程步补充了标准 SAS 回归过程步（如 GLM、REG 和 NLIN）提供的方法。这些过程步可以处理大多数情况下指定的回归模型和已知固定数量参数的模型。但是如果对模型没有预先的了解，或者知道数据不能用具有固定参数数量的模型表示，则可以使用 TPSPLINE 过程步对数据建模。

（96）TRANSREG 变换回归拟合线性模型，可选择平滑、样条、Box-Cox 以及变量的其他非线性变换。使用本过程步可以通过散点图拟合单条曲线或多条曲线，每条曲线对应分类变量的一个级别。常用在其他分析前的试验设计和变量分类。

（97）TREE 绘制聚类分析结果的谱系图，它读取由聚类分析过程步 CLUSTER 或 VARCLUS 过程步创建的数据集生成树状谱系图，该树状结构显示了层次聚类分析的结果。

（98）TTEST 学生 t 检验过程步，计算一个样本、一对观测、两个独立样本和 AB/BA 交叉设计的置信区间。该过程步使用 ODS 图形来创建图形作为其输出的一部分。

（99）VARCLUS R 型聚类分析过程步，采用分解法或迭代法将一组数值分成不重叠的簇，与每个簇关联的是簇中变量的线性组合，可以是主成分或者质心。

（100）VARCOMP 处理具有随机效应的一般线性模型，估计每个随机效应对因变量方差的贡献。随机效应是分类效应，其假定层级是从一个无限可能层级的总体中随机选择的。GLM、MIXED 和 GLIMMIX 也适用于类似的随机效应模型，但对于特定的设计和模型，VARCOMP 计算效率更高。

（101）VARIOGRAM 对二维空间数据计算空间连续性的经验度量，这些度量是样本数据对之间距离的函数。所估计的连续性度量为经验性的半变异函数和协方差。该过程步还提供 Moran I 和 Geary C 空间自相关统计量，以及 Moran 散点图，用于可视化观测周围指定邻域内的空间关联。该过程步使用 ODS 图形来创建图形作为其输出的一部分。

第19章

大数定律与中心极限定理

要了解数据科学首先要深刻理解数据分析科学中的若干重要定律或定理，其中最重要的就是大数定律（Law of Large Number，LLN）和中心极限定理（Central Limit Theorem，CLT），前者探讨样本均值收敛问题，后者探讨极限分布问题。

如果在试验中变量的取值相互不影响，我们称该随机变量是相互独立。如果随机变量相互独立且具有相同的概率分布，则称这组随机变量的分布满足独立同分布（Independent and Identically Distributed，i.i.d）。大数定律告诉我们，在随机过程中，随着试验次数的增加，随机变量的预期值与实际值之间的百分比差会趋向于0。即随着样本数量越大，样本均值越接近期望值。

人们在重复试验中发现事件发生频率一般会趋向于一个稳定值，而现实世界大量使用的算术平均值具有某种稳定性。统计学上的大数定律并不是单个定律，而是一组定律，包括弱大数定律和强大数定律。弱大数定律和强大数定律条件是相同的，即对于独立同分布的随机变量序列 $\{x_1, x_2, \cdots, x_n\}$，有 $\mathrm{E}[X_i]=\mu$。所谓大数定律的强弱，其区别在于定律断言收敛的方式不同，前者是依概率 P 收敛（Convergence In Probability），记为 $\overline{X}_n \xrightarrow{P} \mu$（当 $n\to\infty$）；而后者是依概率1收敛，就是几乎必然收敛（Converge Almost Surely），记为 $\overline{X}_n \xrightarrow{as} \mu$（当 $n\to\infty$）。

19.1 大数定律

19.1.1 弱大数定律

假如 $x_1, x_2, \cdots, x_n$ 为独立同分布的随机变量序列，变量具有相同的均值 $\langle X_i\rangle=\mu$ 和标准差 σ；则变量 $X=\frac{1}{n}\sum_{i=1}^{n}x_i$，当 $n\to\infty$ 时，变量 X 的均值 $\langle X\rangle$ 等于变量的总体均值 μ。即

$$\langle X\rangle=\frac{\langle x_1+\cdots+x_n\rangle}{n}=\frac{1}{n}\left(\langle x_1\rangle+\cdots+\langle x_n\rangle\right)=\frac{1}{n}(n\mu)=\mu$$

且变量 X 的方差 $\mathrm{var}(X)$ 为 $\frac{\sigma^2}{n}$：

$$\mathrm{var}(X)=\mathrm{var}\left(\frac{x_1+\cdots+x_n}{n}\right)=\mathrm{var}\left(\frac{X_1}{n}\right)+\cdots+\mathrm{var}\left(\frac{X_n}{n}\right)=\frac{\sigma^2}{n^2}+\cdots+\frac{\sigma^2}{n^2}=\frac{\sigma^2}{n}$$

根据切比雪夫不等式 $P(|x-\mu|\geqslant k\sigma)\leqslant\frac{1}{k^2}$（其中 k 为标准差的倍数，有 $k>0$），对所有 $\varepsilon>0$ 有如下不等式成立：

$$P(|X-\mu|\geqslant\varepsilon)\leqslant\frac{\operatorname{var}(X)}{\varepsilon^2}=\frac{\sigma^2}{n\varepsilon^2}$$

当 $n\to\infty$ 时，上面的不等式变为 $\lim_{n\to\infty}P(|x-\mu|\geqslant\varepsilon)=0$。也就是说，对任意大于零的正数 ε，平均值 $\left|\frac{x_1+\cdots+x_n}{n}-\mu\right|<\varepsilon$ 的概率在 $n\to\infty$ 时趋向于1。这说明样本均值依概率收敛于期望值。结论我们已经知道，但如何理解这个大数定律呢？首先要从马尔可夫不等式和切比雪夫不等式说起。

对于某个取值大于等于0的随机变量 x，x 取值大于等于某个正数 α 的概率 $P(x\geqslant\alpha)$ 不可能超过随机变量 x 的期望 $\mathbb{E}(x)$ 除以该正数 α；也就是说 x 取值大于某个正数 α 的概率具有固定的上界。这一结论是由著名的马尔可夫不等式给出的，其数学定义和证明如下：

1. 马尔可夫不等式

马尔可夫不等式为：如果 x 为非负数，有 $P(x\geqslant a)\leqslant\frac{\mathbb{E}(x)}{a^2}$。

证明步骤：数学期望计算公式如下：

$$\mathbb{E}(x)=\int_0^\infty xP(x)\mathrm{d}x=\int_0^a xP(x)\mathrm{d}x+\int_a^\infty xP(x)\mathrm{d}x$$

由于 $P(x)$ 是概率密度，因此它肯定大于等于0；且由于 x 是正数，则 x 必然大于等于0，因此 $\int_0^a xP(x)\mathrm{d}x$ 必然大于等于0。上面的等式就演变为如下不等式：

$$\mathbb{E}(x)=\int_0^a xP(x)\mathrm{d}x+\int_a^\infty xP(x)\mathrm{d}x\geqslant\int_a^\infty xP(x)\mathrm{d}x$$

由于 $x\geqslant a$ 则代入上面的不等式进一步演化为如下不等式：

$$\int_a^\infty xP(x)\mathrm{d}x\geqslant\int_a^\infty aP(x)\mathrm{d}x=a\int_a^\infty P(x)\mathrm{d}x=aP(x\geqslant a)$$

从而有结论：$\mathbb{E}(x)\geqslant aP(x\geqslant a)$ 从而推导出了马尔可夫不等式：

$$P(x\geqslant a)\leqslant\frac{\mathbb{E}(x)}{a}$$

这个抽象的马尔可夫不等式在现实中也具有实用价值，比如张三的打靶平均成绩是5环，如果张三跟你说他今天在军训打靶时超常发挥，说自己打靶时3/4都是8环以上，则张三的陈述是否值得相信？

本例中 $a=8$，根据马尔可夫不等式有 $P(x\geqslant 8)$ 的概率应该不可能超过 $\frac{\mathbb{E}(x)}{a}=5/8=0.625=62.5\%$，也就是说根据张三的平均成绩为5环的实际情况，张三打靶8环以上的概率不可能超过62.5%。现在张三自称3/4=75%都是8环以上，由于62.5% < 75%，说明张三是在夸大其词，其陈述是不可信的。

2. 切比雪夫不等式

根据前面的马尔可夫不等式，当 $a=k^2$ 时，代入不等式可得

$$P\left[(x-\mu)^2\geqslant k^2\right]\leqslant\frac{\left\langle(x-\mu)^2\right\rangle}{k^2}=\frac{\sigma^2}{k^2}$$

则对均值为 μ 和方差 σ^2 为随机变量 x，对所有 $k>0$ 都有如下不等式成立：

$$P(|x-\mu|\geqslant k^2)\leqslant\frac{\sigma^2}{k^2}$$

从而推导出切比雪夫不等式：

$$P(|x-\mu| \geqslant k\sigma) \leqslant \frac{1}{k^2}$$

切比雪夫不等式告诉我们越远离平均值 μ 的发生概率是越低的。比如以正态分布为例，数据分布在 2 倍 σ 之外的概率根据切比雪夫不等式有 $P(|x-\mu| \geqslant 2\sigma) \leqslant \frac{1}{2^2}=0.25$，即数据落在 2 倍 σ 之外的概率应该至少小于 25%。

19.1.2　三种大数定律

如果用一句话概括弱大数定律就是：**独立同分布的随机序列，并且期望存在，则样本均值依概率收敛于总体均值**。具体而言存在如下三种表述：

（1）切比雪夫大数定律：设随机变量 $X_1, X_2, \cdots, X_n$ 相互独立，均具有有限方差，且被同一常数 C 所限定；即 $D(X_i) < C$ 对 i=1、2、…、n 成立，则对任意正数 ε，有

$$\lim_{n\to\infty} P\left\{\left|\frac{1}{n}\sum_{i=1}^{n} X_i - E(X_i)\right| < \varepsilon\right\}=1$$

其中当随机变量 $X_1, X_2, \cdots, X_n$ 具有相同的数学期望 $E(X_i)=\mu$ 时，上面的公式变为

$$\lim_{n\to\infty} P\left\{\left|\frac{1}{n}\sum_{i=1}^{n} X_i - \mu\right| < \varepsilon\right\}=1$$

也就是说当 n 趋于无穷大时，样本均值接近于总体均值（期望）。切比雪夫大数定律要求随机变量相互独立，期望和方差都存在，但不要求独立同分布。

（2）伯努利大数定律：假设 μ 是 n 次独立重复试验中事件 A 发生的次数，p 是事件 A 在每次试验中发生的概率 $(0 < p < 1)$，则对任意正数 $\varepsilon > 0$，有

$$\lim_{n\to\infty} P\left\{\left|\frac{\mu}{n}-p\right| < \varepsilon\right\}=1 \quad \text{等价于} \lim_{n\to\infty} P\left\{\left|\frac{\mu}{n}-p\right| \geqslant \varepsilon\right\}=0$$

伯努利大数定律说明当试验次数 n 很大时，事件 A 发生的频率与概率有较大差别的可能性是很小的，也就是说有较小偏差的可能性充分大。因此当试验次数 n 很大时，我们可以用事件发生的频率代替事件发生的概率，从而在数学上证明了频率具有稳定性。伯努利大数定律是从二项分布，且存在相同期望和方差时导出频率等于概率这一结论。

（3）辛钦大数定律：设随机变量 $X_1, X_2, \cdots, X_n$ 是相互独立同分布的随机变量序列，且 $E(X_n)=\mu$，则对于任意正数 ε 有：

$$\lim_{n\to\infty} P\left\{\left|\frac{1}{n}\sum_{i=1}^{n} X_i - \mu\right| < \varepsilon\right\}=1$$

辛钦大数定律是弱大数定律的一种，它要求期望存在（数据存在一阶矩），但没有要求随机变量的方差存在，因此它比伯努利大数定律具有更广泛的应用范围。

弱大数定律的核心是“依概率收敛”，但我们不知道是否随着 n 增大，即使概率很小，样本均值有没有可能偶然偏离总体均值 μ 很多呢？当强大数定律被证明后，我们已经知道样本均值会几乎会处处收敛到总体均值 μ 上。

大数定律简单而直观的描述为：**如果有一个随机变量 X，不断地观察它的 n 个采样**

值 $X_1, X_2, X_3, \cdots, X_n$，然后计算这 n 个采样值的平均值 $\overline{X}_n$，当 n 趋向于正无穷的时候，这个平均值就会收敛于这个随机变量 X 的期望，它是一个常数。

$$\lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^{n}x_i=\mu$$

19.1.3 图形化证明

下面我们做抛硬币试验，理论上每次得到正面和反面的可能性是一样的，即单次抛硬币试验是取值为 0 或 1 的离散均匀分布。然而，现在我们不是抛一次硬币，而是把重复独立抛 n 次硬币算作一次试验，这样我们在一次试验中就获得了 n 个采样值：$X_1, X_2, \cdots, X_n$，然后统计 n 次试验中硬币国徽朝上的次数，即国徽朝上事件成功的次数 X。当这个试验 n 次数趋向于无穷时，我们已可以预见国徽朝上次数的平均值收敛于随机变量 X 的数学期望。当然也可换一种说法，即在一个盒子里包含 n=100 枚硬币，我们每次摇一摇盒子，然后清点其中国徽朝上的硬币数量 X。我们连续摇 10000 次，则我们可以预见其中国徽朝上的硬币数将会收敛于 $n/2$=50。程序 19-1 用 SAS 代码可证明这一点：

程序19-1　重复多次抛n枚硬币试验其均值趋向于n/2

```
data sample;
  n=100;
  xsum=0;
  do i=1 to 100000;/*做100000次试验*/
    x=0; /*x 记录一次抛100枚硬币试验中朝上的个数*/
    do j = 1 to n;
      v=RAND('UNIFORM');/*连续均匀分布生成随机数v>0.5概率*/
      if v>0.5 then v=1;
      else v=0;
      x=x+v;
    end;

    xsum= xsum + x;
    y= xsum/i; /*当试验次数足够多时，此值收敛于n/2*/
    output;
  end;
  put n= y=;
  keep x y i;
run;
```

将试验中硬币国徽朝上的个数 x 以及历次试验国徽朝上的个数的平均数 y 作直方图（见程序 19-2）。当试验次数足够时可以发现，其中国徽朝上的个数平均值将一枝独秀，即均值落在 50 上的次数越来越多，相比之下落在其他值上的次数就非常微不足道（见图 19-1），它说明大数定律作用下均值逼近期望的过程。

程序19-2　显示成功事件发生次数 x 及其均值 y 的直方图分布，呈现一枝独秀

```
proc sgplot data=sample;
  title 'Histrogram';
  histogram x;
  histogram y;
  keylegend / location=inside position=topright across=1;
  xaxis display=(nolabel);
run;
```

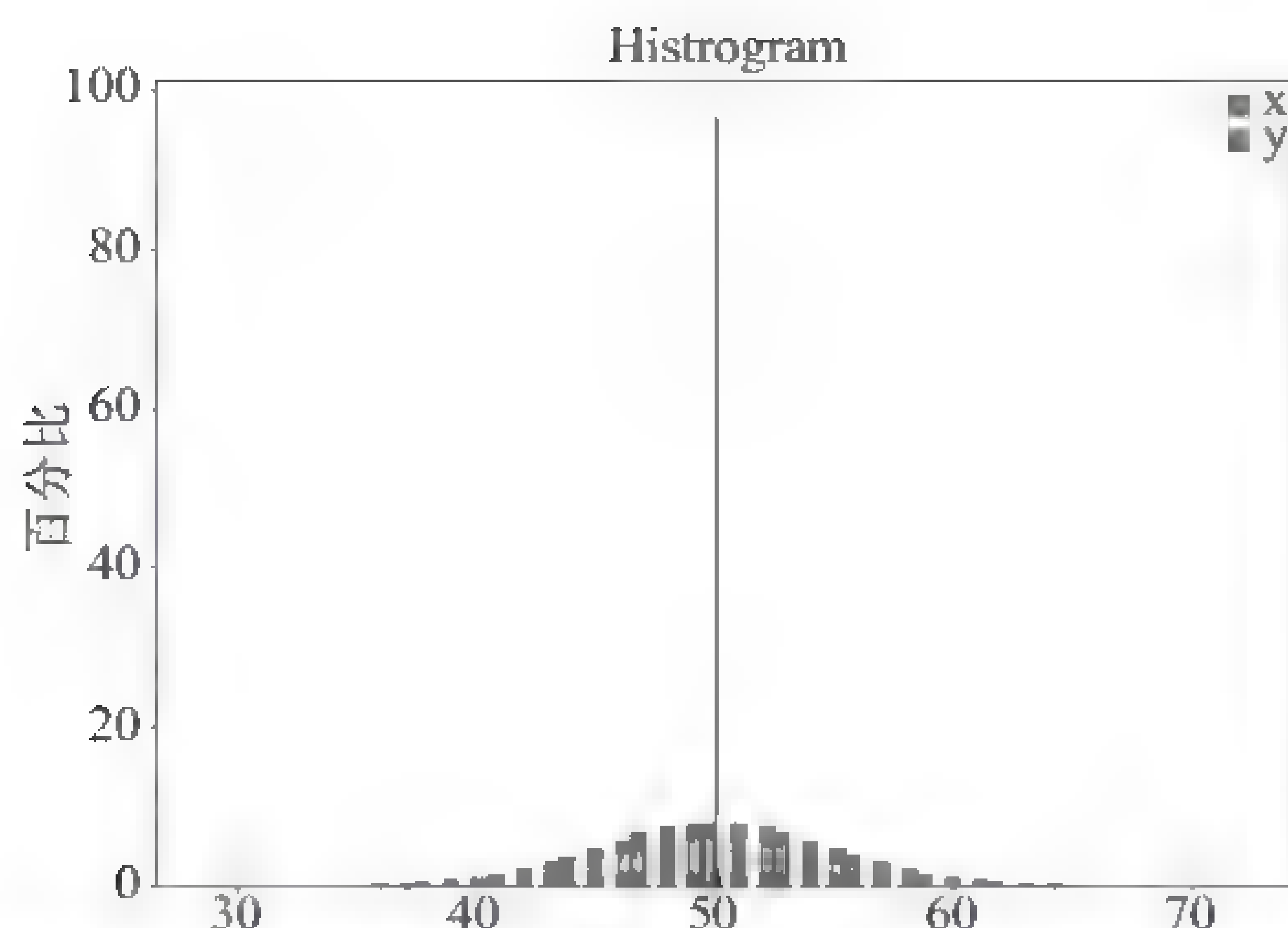


图 19-1　数据量较大时均值的频数不断增长（集中）

我们也可以换一种视角来观察试验的结果（见程序 19-3），比如以试验过程中国徽朝上的个数的平均数 y 为纵坐标，以试验次数为横坐标，则可以看到随着试验次数 i 的增长，均值无限逼近期望值 50 的过程（见图 19-2）。

程序19-3　成功事件发生次数均值与试验次数的关系，大数收敛

```
proc sgplot data=sample;
  title 'Mean/N Plot';
  series x=i y=y ;
run;
```

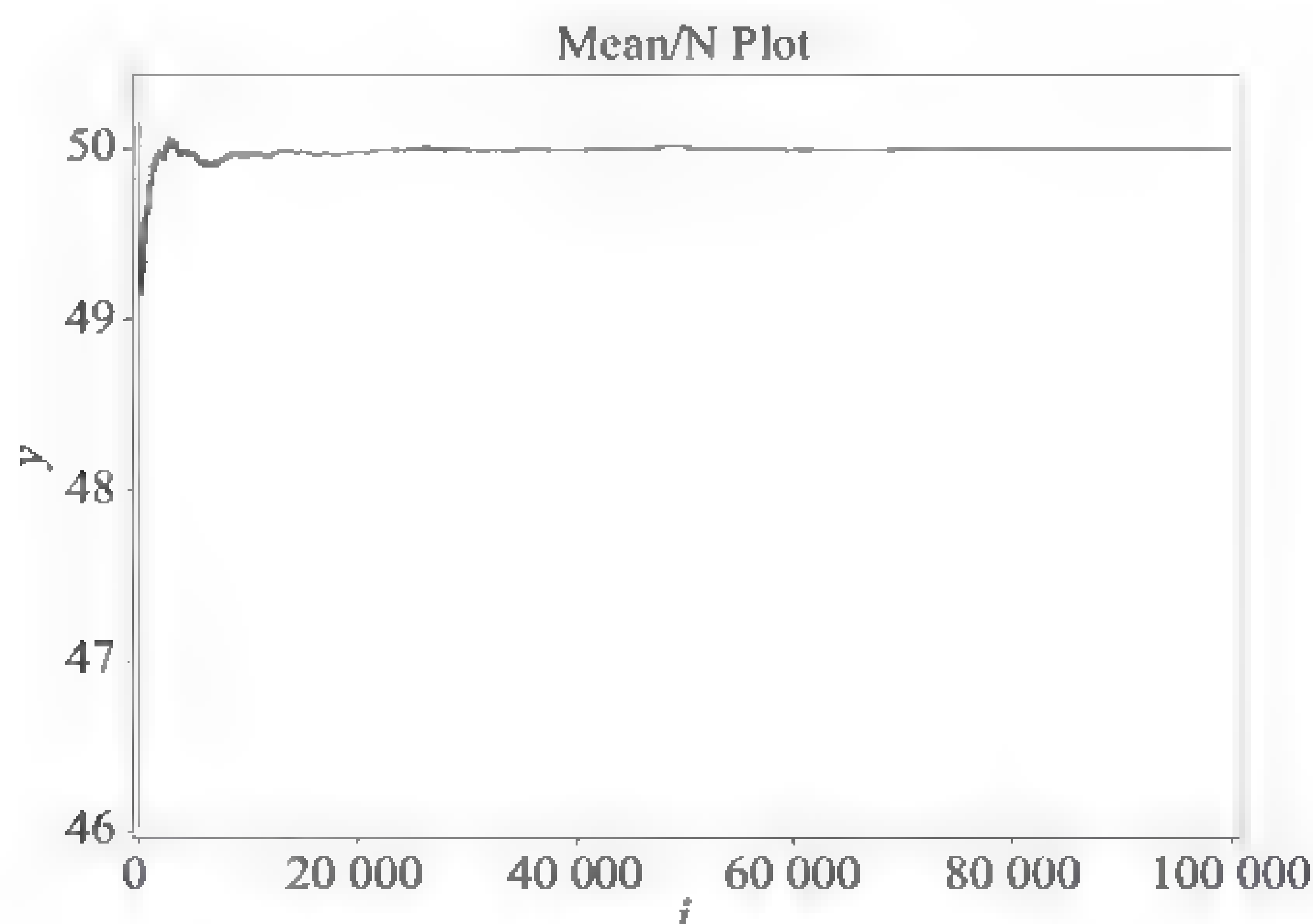


图 19-2　均值与试验次数的关系

从图 19-2 中可以看出，一次试验中抛 100 次硬币（或同时抛 100 枚硬币），朝上次数的平均值会随着试验次数 n 的增大不断地收敛，其期望为 50。我们绘制试验次数和国徽朝上次数的平均值的关系，可以看到随着试验次数的增加，样本均值会依概率收敛于 50。实际上，样本均值依概率收敛于总体均值是一个普遍的数学规律，用均匀分布的抛骰子试验也可证明这一点，图 19-3 就是用 SAS 语言模拟 12 个抛骰子试验，单个试验中连续抛 3000 次，可以证明抛出来的点数的均值都收敛于期望 3.5。

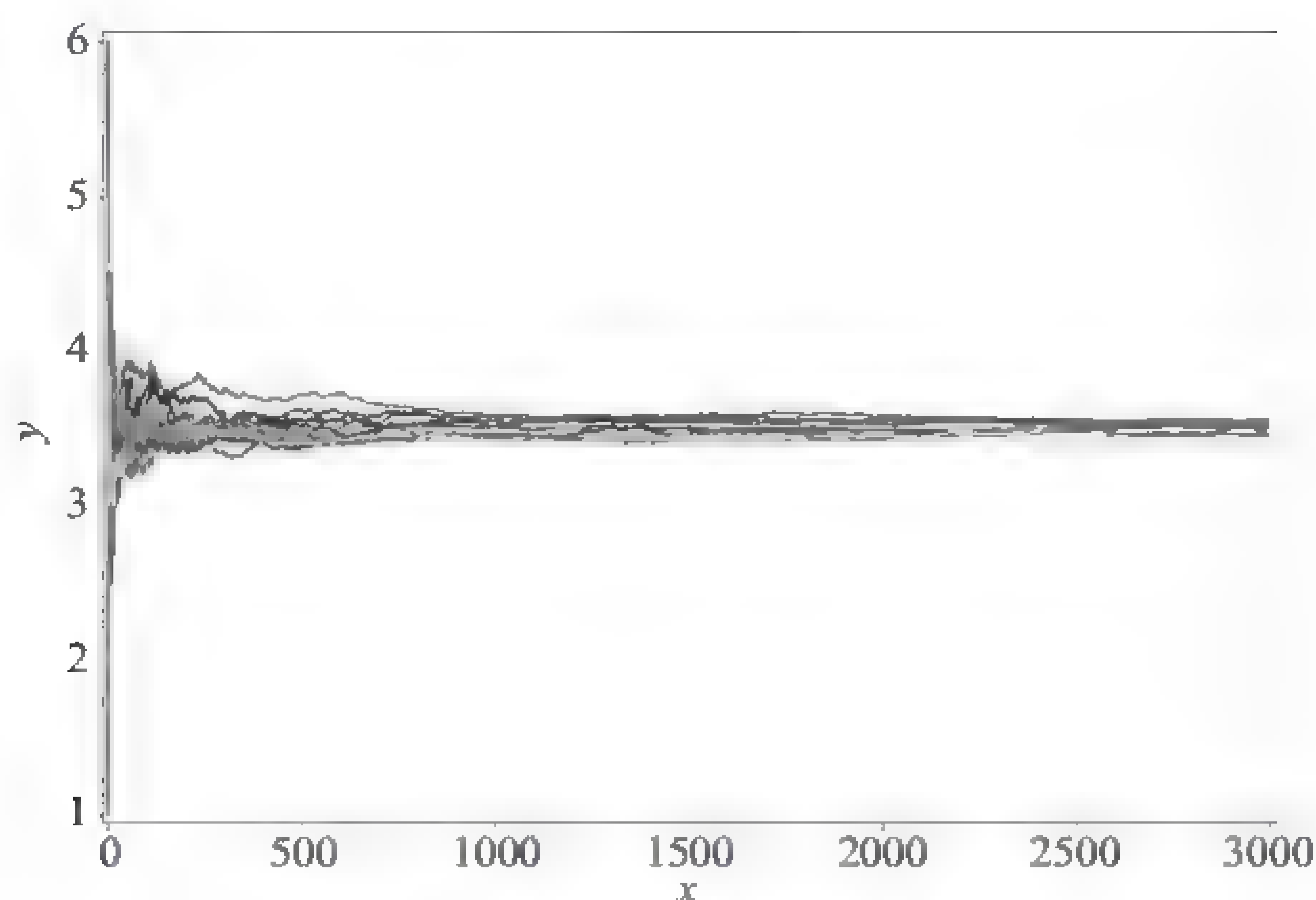


图 19-3　抛 12 枚骰子均值收敛情况

至于伯努利大数定律，频率 n 在很大时接近概率也是非常直观的。比如抛一次硬币时获得国徽朝上和朝下的可能性是一样的，它们遵循的是离散均匀分布，即 Uniform{0, 1} 的等概率分布，同时也是 Bernoulli(p=0.5) 的伯努利分布。图 19-4 就是抛 100,000 次时点数在 0 和 1 两个取值上接近概率 0.5（两个频数直方图等高，而总概率为 1）。

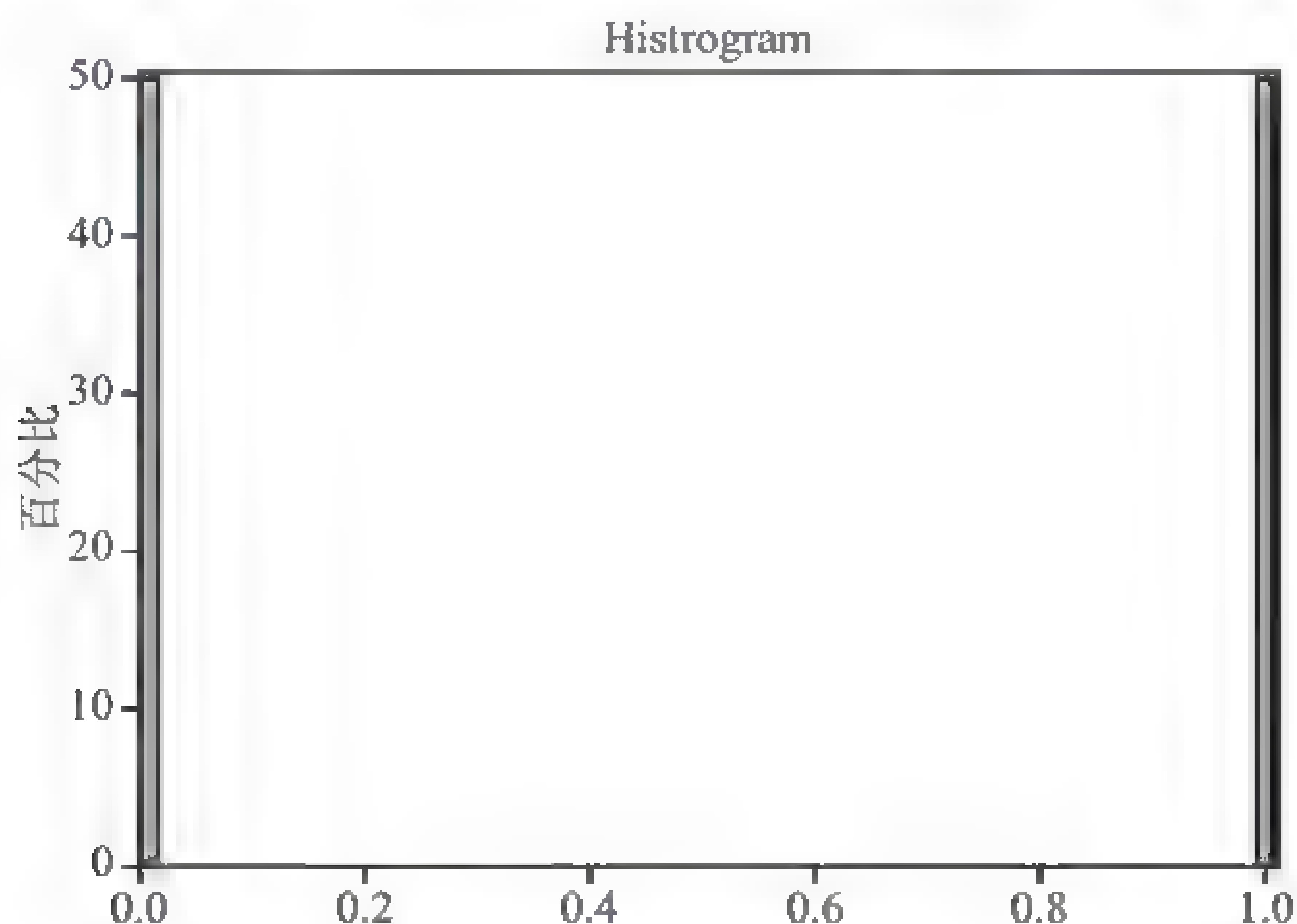


图 19-4　频率在试验次数很大时接近于概率

输出图 19-4 结果对应的 SAS 代码如程序 19-4 所示。

程序19-4　等概率抛硬币试验揭示的大数定律

```
data sample;
  n=100000; /*抛100000次硬币，看国徽朝上和朝下的情况是均等的*/
  do i=1 to  n;
    x=RAND('UNIFORM'); /* 0 < x < 1  */
    if x>0.5 then x=1;
    else x=0;
    output;
```

```
  end;
  keep x;
run;

proc sgplot data=sample;
  title 'Histrogram';
  histogram x / binwidth=2;
  keylegend / location=inside position=topright across=1;
  xaxis max=1 display=(nolabel);
run;
```

我们观察试验次数的增加，硬币朝上次数的平均值会如何收敛，结果也发现收敛到 0.5。频数具有稳定性收敛，事件发生的频率可以代表事件发生的概率。对于抛硬币试验，程序 19-5 利用伯努利试验也可得到相同的试验结果，其输出如图 19-5 所示。

程序19-5　伯努利试验揭示的大数定律

```
data sample;
  n=10000;
  xsum=0;
  do i = 1 to n;
    x=RAND('BERNOULI',0.5); /*试验成功返回 1 */
    xsum=xsum+x;
    x=xsum/i;
    output;
  end;
  keep i x ;
run;

proc sgplot data=sample;
  series x=i y=x ;
  yaxis min=0 max=1;
run;
```

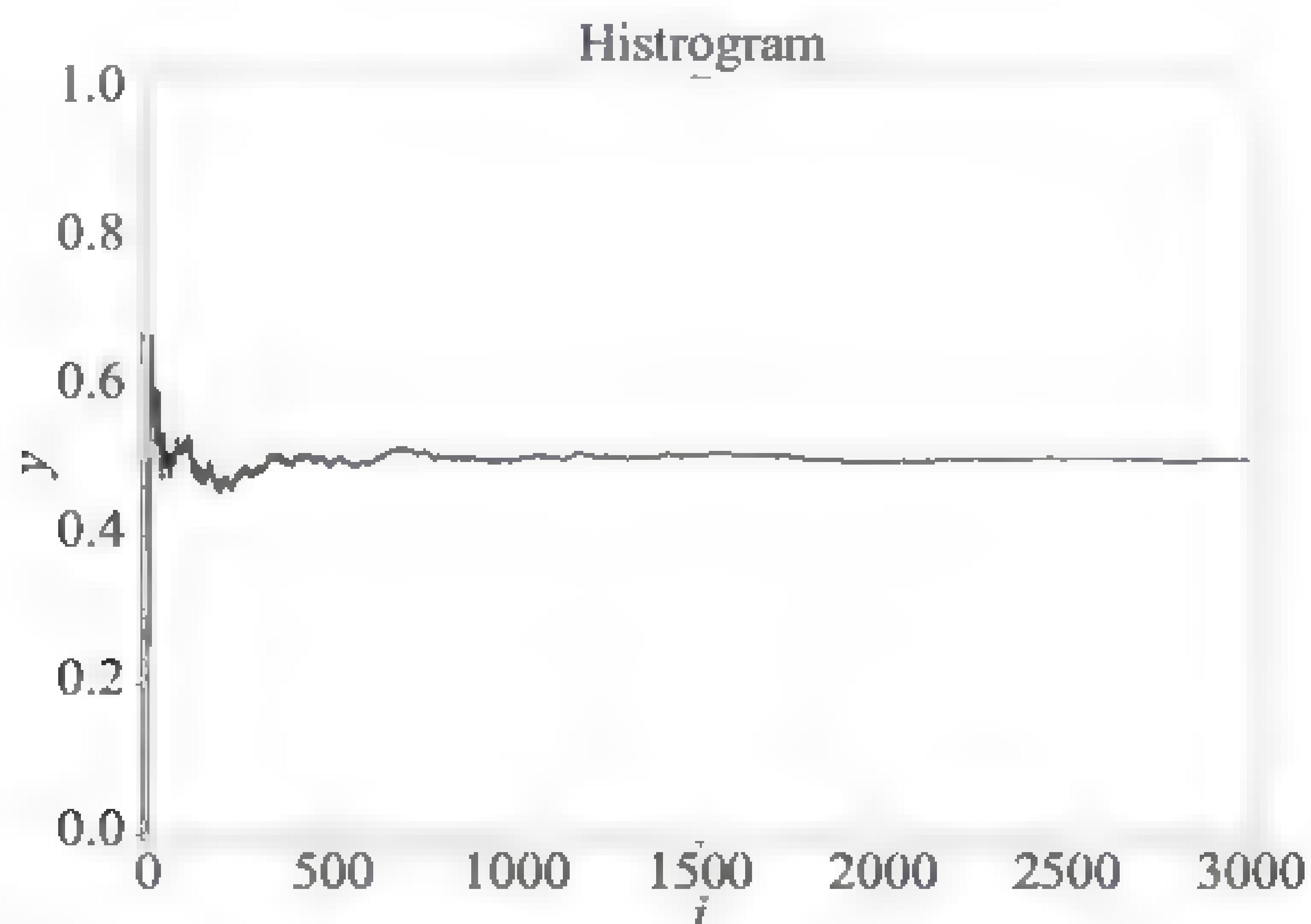


图 19-5　伯努利试验揭示大数条件下事件发生的频率可代表概率

19.1.4　强大数定律

概括而言，对于独立同分布的随机序列，并且期望存在，若样本均值以概率为 1 收敛于总体均值，也就是说几乎必然收敛（Almost Surely Convergence）于总体均值，则称

该随机序列服从强大数定律。1928 年柯尔莫哥洛夫找到了独立同分布下强大数定律的充要条件，它告诉我们随着 n 的增大，样本均值几乎处处收敛于总体均值。

19.2 中心极限定理

中心极限定理指出：**大量相互独立的随机变量，其均值的分布以正态分布为极限。**中心极限定理也并不是一个而是一组定理。包括棣莫弗 - 拉普拉斯定理、林德伯格 - 列维定理、林德伯格 - 费勒定理等。中心极限定理的重要意义在于，根据定理的结论其他概率分布可用正态分布作为近似。比如二项分布在 Bin(n, p) 相当大时，p 接近于 0.5 时近似 $N(np, np(1-p))$，泊松分布 $Po(\lambda)$ 当取样样本数很大时近似 $N(\lambda, \lambda)$ 分布。

最早的中心极限定理是法国数学家棣莫弗在 1733 年发现并由拉普拉斯进行扩展形成的，但更一般的随机变量服从中心极限定理则是 1901 年由俄国数学家里雅普诺夫进行精确证明。中心极限定理包括如下几种形式：

（1）棣莫弗 - 拉普拉斯定理：服从二项分布 Bin(n, p) 的随机变量序列，其分布以均值为 np，方差为 $np(1-p)$ 的正态分布为极限。这解释了高尔顿钉板试验中小球下落累积的高度曲线服从钟形曲线。高尔顿钉板是生物统计学家弗朗西斯 • 高尔顿发明用 n 排钉子模拟 N 重伯努利试验验证中心极限定理的试验装置。

设随机变量 X_n 为具有参数 $n, p\ (0 < p < 1)$ 的二项分布，则对于任意实数 x 有

$$\lim_{n\to\infty} P\left\{\frac{X_n - np}{\sqrt{np(1-p)}} \leqslant x\right\} = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{\frac{-t^2}{2}dt}$$

（2）林德伯格 - 列维定理：独立同分布且具有有限数学期望和方差的随机变量序列，其标准化和以标准正态分布为极限。它是棣莫弗 - 拉普拉斯定理的扩展，该定理也称为独立同分布的中心极限定理。

设随机变量 $X_1, X_2, \cdots, X_n$ 相互独立，服从同一分布，且具有相同的数学期望和方差：$E(X_k)=\mu$，$D(X_k)=\sigma^2 \neq 0\ (k=1, 2, \cdots, n)$，则随机变量 $Y_n = \dfrac{\sum_{k=1}^{n} X_k - n\mu}{\sqrt{n}\sigma}$ 的分布函数 $F_n(X)$ 对任意实数 x 有

$$\lim_{n\to\infty} F_n(x) = \lim_{n\to\infty} P\left\{\frac{\sum_{k=1}^{n} X_k - n\mu}{\sqrt{n}\sigma} \leqslant x\right\} = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{\frac{-t^2}{2}dt}$$

（3）林德伯格 - 费勒定理：在满足一定条件时，独立但不同分布的随机变量序列的“标准化和”依然以标准正态分布为极限。该条件包括：

- 随机变量序列为独立但不同分布，期望$E[X_i]=0$且具有有限方差
- 如果对每个$\varepsilon > 0$，序列满足如下林德伯格条件：$\lim_{n\to\infty} P\frac{1}{\sigma_n^2}\sum_{i=1}^{n} E\left[X_i^2; \{|X_i| > \epsilon\sigma_n\}\right] = 0$，

  假定n个独立随机变量$X_1, X_2, \cdots, X_n$，每一个X_i服从任意概率分布$P(x_1, x_2, \cdots, x_n)$，

它具有有限的均值μ_i和方差 σ_i^2，则归一化变量$X_{\text{norm}}=\dfrac{\sum_{i=1}^{n}x_i-\sum_{i=1}^{n}\mu_i}{\sqrt{\sum_{i=1}^{n}\sigma_i^2}}$具有逼近正态分布的极限累积分布函数（CDF）。若未做归一化处理，则变量$X=\dfrac{1}{n}\sum_{i=1}^{n}x_i$服从均值为$\mu_X=\mu_x$、方差为 $\sigma_X=\dfrac{\sigma_x}{\sqrt{n}}$的正态分布。本定理是林德伯格-列维定理的扩展，是中心极限定理的高级形式。

虽然中心极限定理有多种形式的表述，但它们共同指出受许多小而无关的随机效应影响的数据会呈现近似正态分布。中心极限定理的核心思想是：**如果相互独立的随机变量服从具有有限均值和方差的任意分布，随机变量的均值趋于正态分布**。这一点无论随机变量服从什么统计分布，它都满足这个定理。

19.2.1　大数定律与中心极限定理关系

大数定律和中心极限定理共同构成了统计分析的基本理论基础。独立同分布的随机事件，在观测次数足够大时，事件的频率会趋于一个稳定的概率值（期望值），这是由大数定律（LLN）所描述的。而事件的分布则趋近于一个稳定的正态分布，这是中心极限定理（CLT）所描述的；该正态分布的期望值就是大数定律中的那个概率值。它们都用来描述独立同分布随机变量和的渐进表现，但它们是在不同收敛速率和前提条件下，大数定律说的是依概率收敛或者几乎必然收敛于某个常数，而中心极限定理说的是按分布收敛，具有更深刻的含义。图 19-6 展示了这些定律和定理的核心内容。

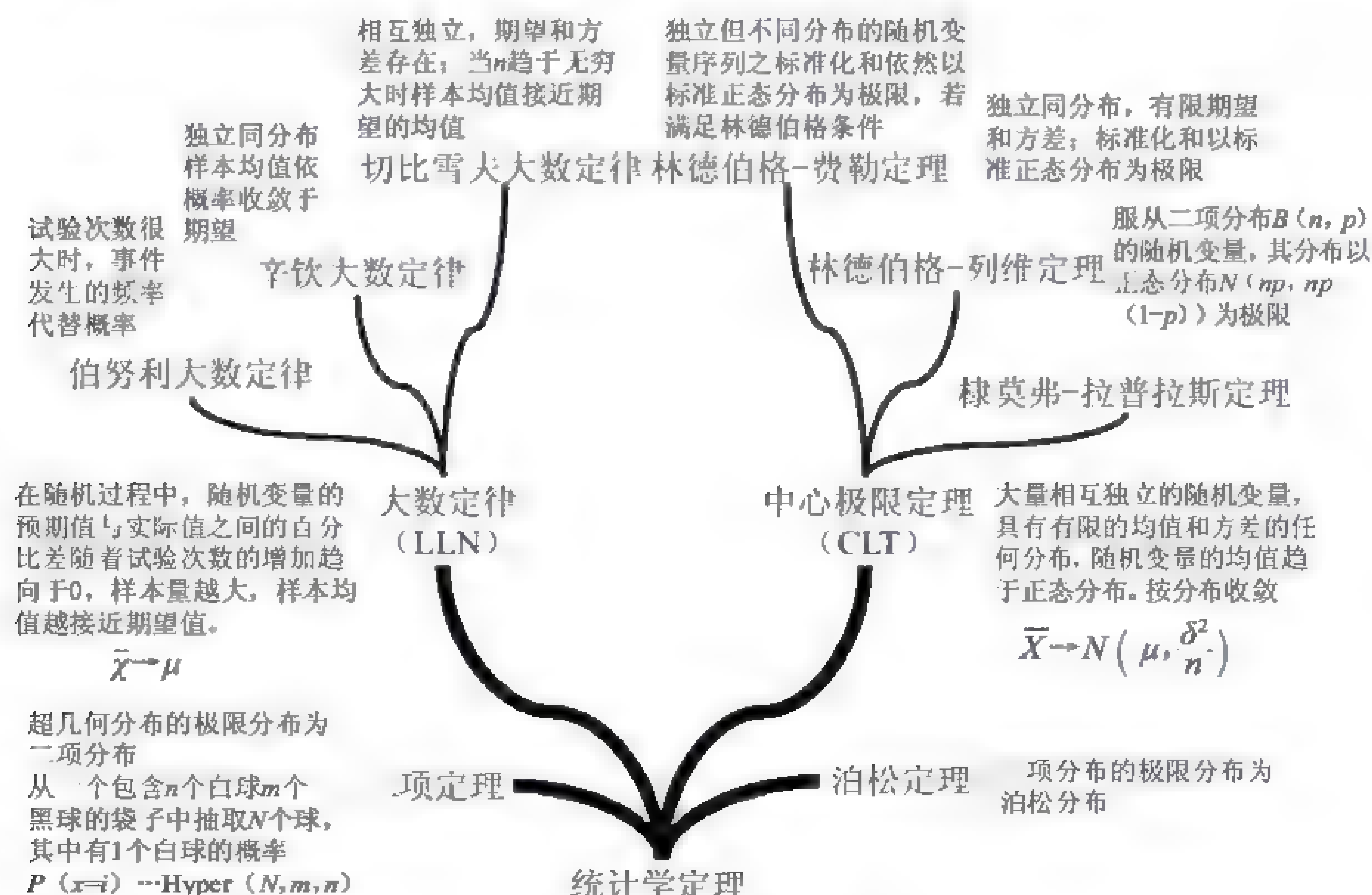


图 19-6　主要统计学定律 / 定理的关系

而表 19-1 展示了大数定律与中心极限定理的关系。

表 19-1　大数定律与中心极限定理的关系

目的	**大数定律** 揭示了大量随机变量的平均结果			**中心极限定理** 揭示大量随机变量的均值分布情况
收敛形式	按概率收敛 p	几乎确定收敛 as		按分布收敛 （分布函数的收敛）
前提条件	独立同分布的随机变量		只需相互独立分布，但要求随机变量的期望和方差均存在	
核心内容	**弱大数定律** 样本数 n 趋于无穷时，样本均值按概率收敛于期望。但并未排除偶尔大的偏差	**强大数定律** 样本数 n 趋于无穷时，样本均值几乎确定收敛于期望。均值不可能偏离太多	**切比雪夫大数定律** 位于其平均数 k 个标准差范围内的比例总是至少为 $1-1/k^2(1<k)$。在 k=2,3,5 时，至少有 3/4=75%、8/9=88.9% 和 24/25=96% 的数据位于 2,3,5 个标准差范围内	**中心极限定理** 样本均值按分布趋向于正态分布。说明的是在一定条件下，大量独立随机变量的平均数是以正态分布为极限。
	辛钦大数定律 不要求随机变量的方差存在，需要独立同分布的条件		**切比雪夫大数定律**是切比雪夫不等式的推论，是大数定律的一般公式。它由切比雪夫不等式 $P\{\|x-EX\|<\varepsilon\} \geqslant 1-DX/\varepsilon^2$ 导出在区间 $(x\pm k\sigma)$ 上的概率	**林德伯格 - 列维定理** 独立同分布的大数定律 **林德伯格 - 费勒定理** 独立但不同分布的随机变量序列的标准化和服从标准正态分布
二项分布	**伯努利大数定律** 试验次数很大时，可以用事件发生的频率来代替事件的概率			**棣莫弗 - 拉普拉斯中心** 二项分布的极限分布是正态分布，说明实践中可用大样本近似处理

19.2.2　图形化证明

下面我们还是用前面的抛硬币来做例子（见程序 19-6），不过这次做 100000 次试验，单次试验中抛 3000 枚硬币或重复抛 3000 次硬币，检查其国徽朝上的枚数 x，或者朝上枚数的平均值 $x/3000$。然后用 PROC SGPLOT 来查看对应的直方图、概率密度曲线以及正态分布曲线。通过对比可以发现其分布确实逼近正态分布曲线（见图 19-7）。

程序19-6　抛硬币试验图形化证明中心极限定律

```
data sample;
  do i=1 to 100000; /*做100000次试验*/
    x=0;
    n=3000;
    do j = 1 to n; /*一次试验中抛 3000 次硬币，看朝上的次数*/
      xx=RAND('UNIFORM'); /* 均匀分布：0 < x < 1  */
      if xx>0.5 then x=x+1;
    end;
    y= x/n; /*当 n 枚数足够多时，其平均值 y 或总和 x 的分布逼近正态分布*/
    output;
  end;
  keep x y ;
run;

proc sgplot data=sample;
  title 'Histrogram';
  histogram y  ;
  density y / type=normal legendlabel='Normal'
    lineattrs=(pattern=solid color=red);
  density y / type=kernel legendlabel='Kernel'
    lineattrs=(pattern=solid color=blue);
  keylegend / location=inside position=topright across=1;
  xaxis display=(nolabel);
run;
```

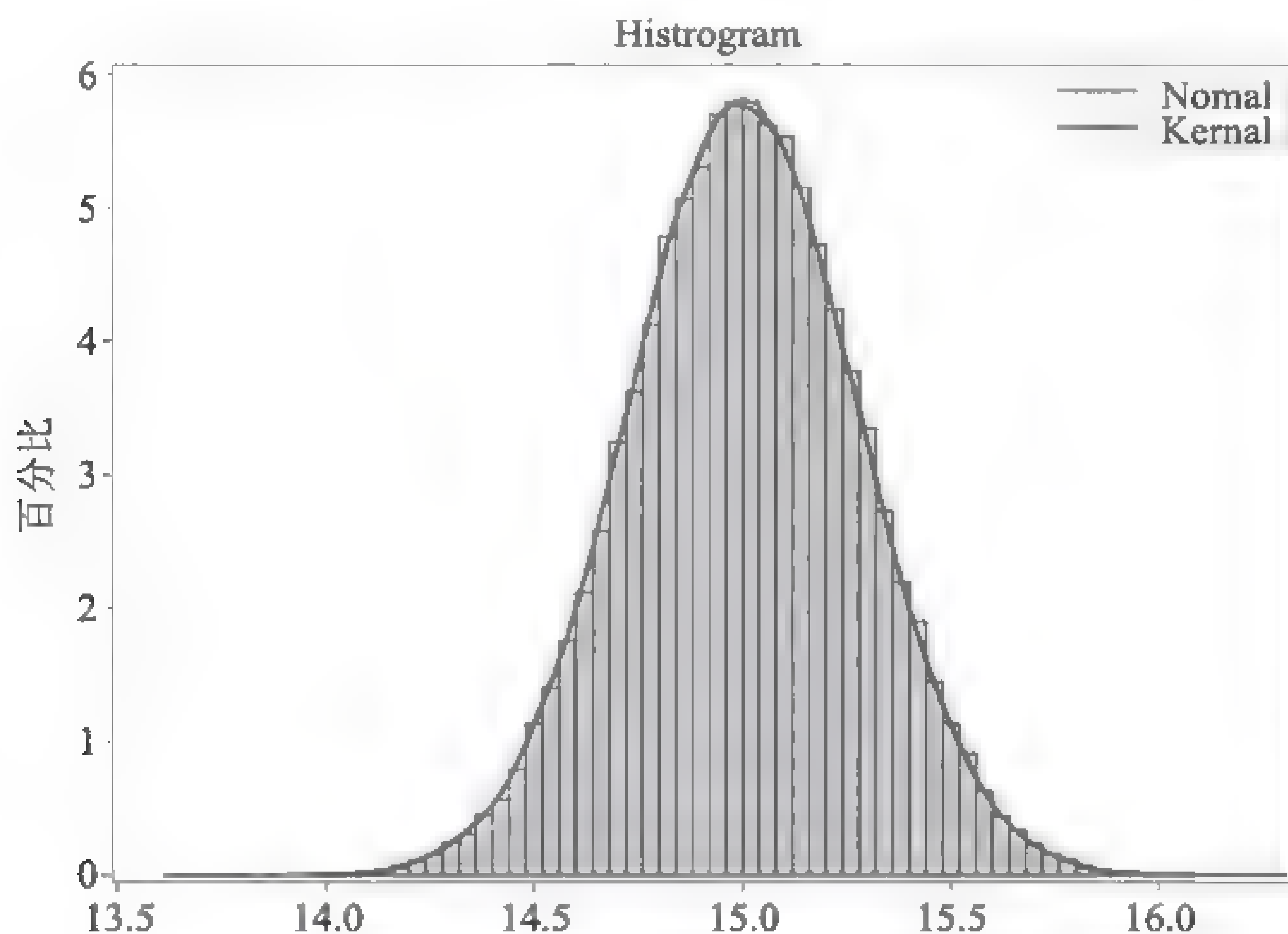


图 19-7　重复试验的均值分布逼近正态分布

在上面的试验中，获得硬币正反面的概率是 0.5，单次试验中的重复次数为 $n=3000$，该问题的实质就是二项分布 Binomial ($n=3000, p=0.5$) 问题。所以当重复次数 n 很大时，棣莫弗 - 拉普拉斯中心极限定理证明二项分布近似于正态分布。这个抛硬币试验在 SAS 里也可以直接使用 SAS 二项分布 Binomial(n, p) 进行模拟，其中 $n=3000$ 时已经很好地逼近正态分布。见程序 19-7 及其输出图 19-8 所示。

程序19-7　棣莫弗-拉普拉斯中心极限定理的图形化证明

```
data sample;
  do i = 1 to 100000;
    p=0.5; n=3000;/*n=1 n=2 n=30 有锯齿 n=3000 完美正态分布*/
    x=RAND('BINOMIAL', p, n); /*n=1,2,...*/
    output;
  end;
  keep x;
run;
proc sgplot data=sample;
  title 'BINOMIAL Distribution (n=3000, p=0.5 )';
  histogram x ;

  density x / type=normal legendlabel='Normal'
    lineattrs=(pattern=solid color=red);
  density x / type=kernel legendlabel='BINOMIAL'
    lineattrs=(pattern=solid color=blue);
  keylegend / location=inside position=topright across=1;
  xaxis display=(nolabel);
run;
```

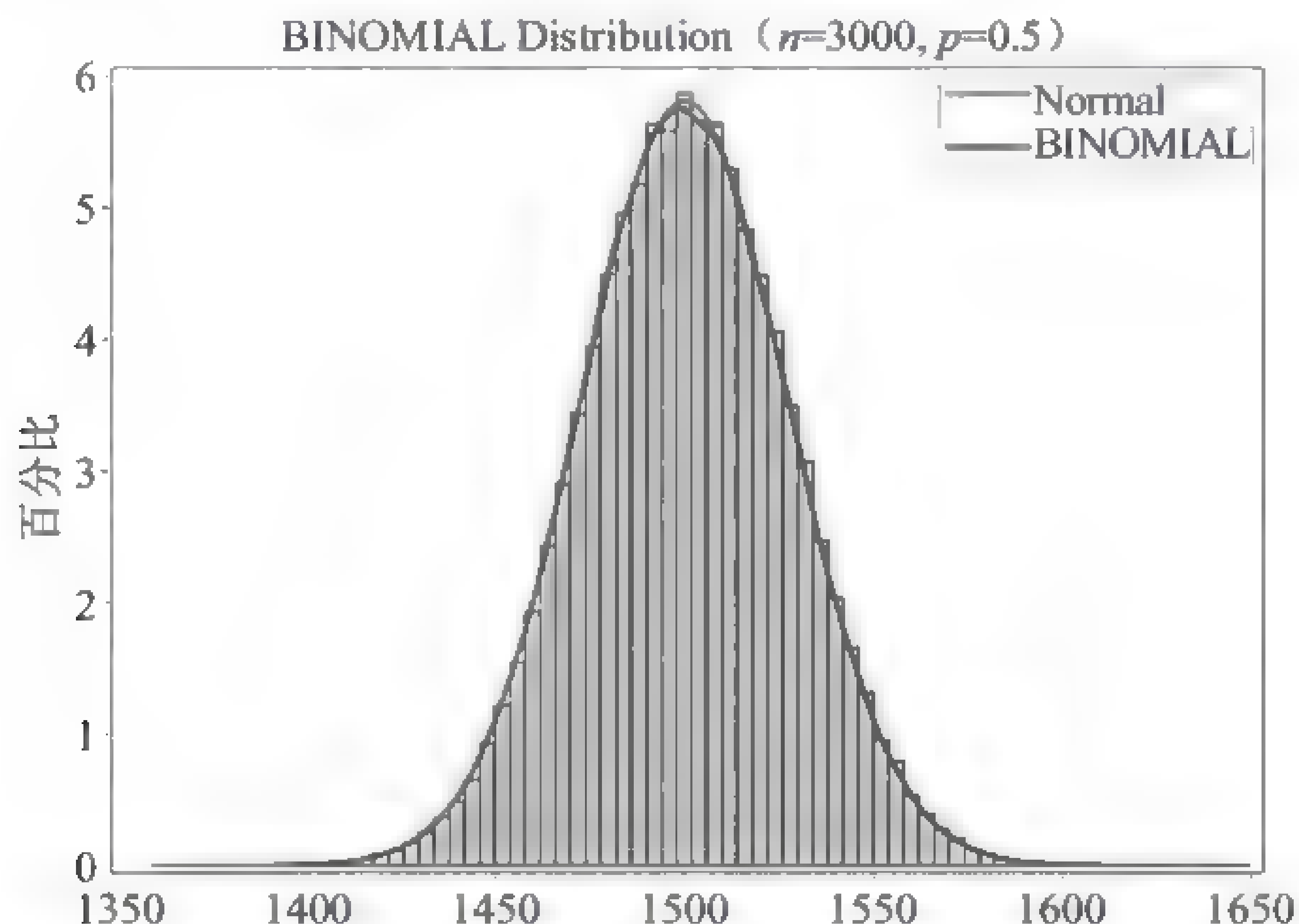


图 19-8　二项分布近似于正态分布（棣莫弗 - 拉普拉斯中心极限定理）

另外，也可以使用抛骰子来证明这一点。我们已知获得骰子 6 个面的可能性也是相等的（1/6），遵循离散均匀分布。中心极限定理并不要求数据的分布类型，而是要求是有限方差即可。为此，我们编写模拟抛骰子的 SAS 代码如程序 19-8 所示，验证当 n 较大时是否服从正态分布，特别注意其均值逼近 3.5，而方差逼近于 0（见图 19-9）。

程序19-8　抛骰子试验证明中心极限定理

```
data sample;
  do i=1 to 10000;
    psum=0;
    n=3000;/*同时抛 3000 枚骰子,记录它们的点数*/
    do j = 1 to n;
      p=floor(RAND('UNIFORM')*6)+1 ; /* 0 < x < 1  */
      psum=psum+p;
    end;
    x=psum/n;
    output;
```

```
  end;
  keep x ;
run;

proc sgplot data=sample;
  title 'UNIFORM Distribution ';
  histogram x      ;

  density x  / type=normal legendlabel='Normal'
    lineattrs=(pattern=solid color=red);
  density x  / type=kernel legendlabel='UNIFORM'
    lineattrs=(pattern=solid color=blue);
  keylegend / location=inside position=topright across=1;
  xaxis display=(nolabel);
run;
```

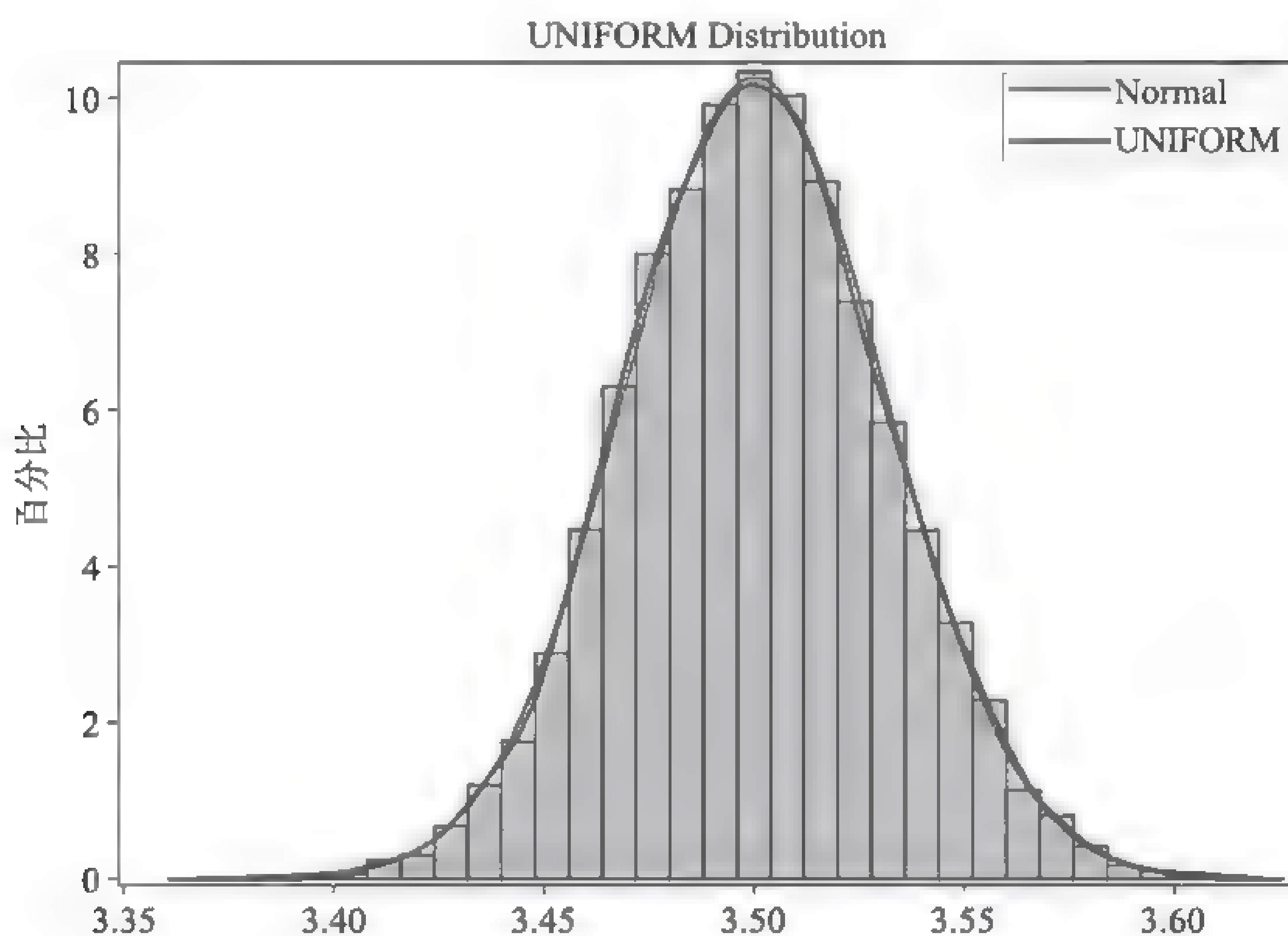


图 19-9　抛骰子证明中心极限定理

从上图中可以看出，其核密度分布曲线和正态分布曲线非常贴合，说明样本的统计分布是逼近正态分布的。通过检查其均值和方差（见程序 19-9）逼近于 3.5 和 0 也可说明这一点。

```
程序19-9　均值与方差接近于期望3.5 和0
proc means data=sample mean var skewness kurtosis;
run;
```

分析变量：x			
均值	方差	偏度	峰度
3.5001849	0.000966660	0.0234725	0.0652270

图 19-10　均值逼近 3.5 且方差逼近于 0

19.2.3 实际用途

在解决现实问题中，棣莫弗 - 拉普拉斯定理是否也有什么实用价值？实际上，该定理的断言可以求解概率近似值。比如某保险公司多年的统计资料表明，理赔用户中由于被盗而向保险公司索赔占比 20%，现在随机抽查 100 个理赔用户，请问因被盗而向保险公司索赔的户数不少于 14 户，且不多于 30 的概率近似值是多少。

这种问题的求解首先是保险公司是否因被盗而对用户进行赔偿，其赔偿户数 X 服从二项分布即 $X\sim B(n, p)$。也就是 $X\sim B(100, 0.2)$ 的二项分布。根据二项分布的期望和方差公式：期望 $E(X)=np=20$，方差 $D(X)=np(1-p)=100\times0.2\times0.8=16$。

根据棣莫弗 - 拉普拉斯定理：二项分布的极限形式为正态分布，因此我们就可用正态分布 $N(\mu, \sigma^2)=N(20, 16)$ 来逼近。于是问题就转化为计算索赔用户数在 14~30 之间的概率近似值，即求 $P\{14\leqslant X\leqslant 30\}$，首先需要将正态分布变换为标准正态分布：

$$P(14\leqslant X\leqslant 30)=P\left(\frac{14-\mu}{\sigma}\leqslant\frac{X-\mu}{\sigma}\leqslant\frac{30-\mu}{\sigma}\right)$$

将 $\mu=20$，$\sigma=4$ 代入，得到 $P(14\leqslant X\leqslant 30)=P\left(-1.5\leqslant\frac{X-20}{4}\leqslant 2.5\right)$，此时查标准正态分布表（参见附录 3：标准正态分布累积概率表）可得：

$$P(14\leqslant X\leqslant 30)\approx\phi(2.5)-\phi(-1.5)=\phi(2.5)-(1-\phi(1.5))=0.9938-(1-0.9932)=0.9269$$

也就是说，随机抽查 100 个索赔用户，因被盗而索赔户数在 14~30 户的概率应该是 92.69%。这个计算结果也可在 SAS 中直接计算求得，如程序 19-10 所示。

程序19-10 棣莫弗-拉普拉斯定理的现实意义

```
data _null_;
  /*服从二项分布：B(n,p),分别计算其期望 EX 和方差 D(X)*/
  n=100; p=0.2;
  x0=14; x1=30;
  /*用正态分布 N(np, np(1-p) 来逼近 - 棣莫弗-拉普拉斯定理*/
  u=n*p;
  dx=n*p*(1-p);
  sx0=(x0-u)/sqrt(dx);
  sx1=(x1-u)/sqrt(dx);

  p_x1=cdf("Normal", sx1);
  p_x0=cdf("Normal", sx0);

  p_x0x1=p_x1- p_x0;
  put "正态逼近 p=" p_x0x1 6.4 "(" p_x0 6.4 ", " p_x1 6.4 ")";

  /*用二项分布直接计算也是可以的*/
  bp_x0=cdf("Binomial", x0, p, n);
  bp_x1=cdf("Binomial", x1, p, n);
  bp_x0x1=bp_x1- bp_x0;
  put "二项分布 p=" bp_x0x1 6.4 "(" bp_x0 6.4 ", " bp_x1 6.4 ")";
run;
```

运行上面的程序，系统输出如下：

```
正态逼近 p=0.9270(0.0668, 0.9938)
二项分布 p=0.9135(0.0804, 0.9939)
```

理解数据分析的真谛应深刻理解大数定律和中心极限定理，另外还有两个有关基础分布的断言需要特别关注，其中一个断言是超几何分布的极限分布是二项分布，即二项定理；另一个断言是二项分布的极限分布是泊松分布，即泊松定理。关于统计分布详情可参阅第 20 章有关内容。

第20章

统计分布

统计分布研究的是特定变量的观测值在分布形态上的特征。定量变量可以分为离散变量和连续变量两大类，因此数据分布也可以分为离散型分布和连续型分布两大类别，其中最重要的统计分布是正态分布，但各种统计分布在特定条件下又紧密相关，从而构成一个衡量不确定性中所蕴含确定性的完整逻辑网络。各种统计分布的存在表明，上帝与你一起抛骰子时也得好好思考一下如何不被你识破！下面从最直观的统计分布“均匀分布”开始讲起。

20.1 均匀分布

均匀分布（Uniform Distribution）也称矩形分布，因为该分布具有等概率特性，所以又称等概分布。均匀分布的变量既可以是连续变量（见图20-1上），也可以是离散变量（见图20-1下）。比如均匀分布在区间 $[a, b]$ 之间的连续变量，其分布称为连续均匀分布，记为 Unif(a, b)。而均匀分布在区间 $[a, b]$ 之间的离散变量，其分布称为离散均匀分布，记为 Unif $\{a, \cdots, b\}$ 。两者的概率密度曲线 $P(x)$ 和累积分布曲线 $D(x)$ 分别如图20-1所示。

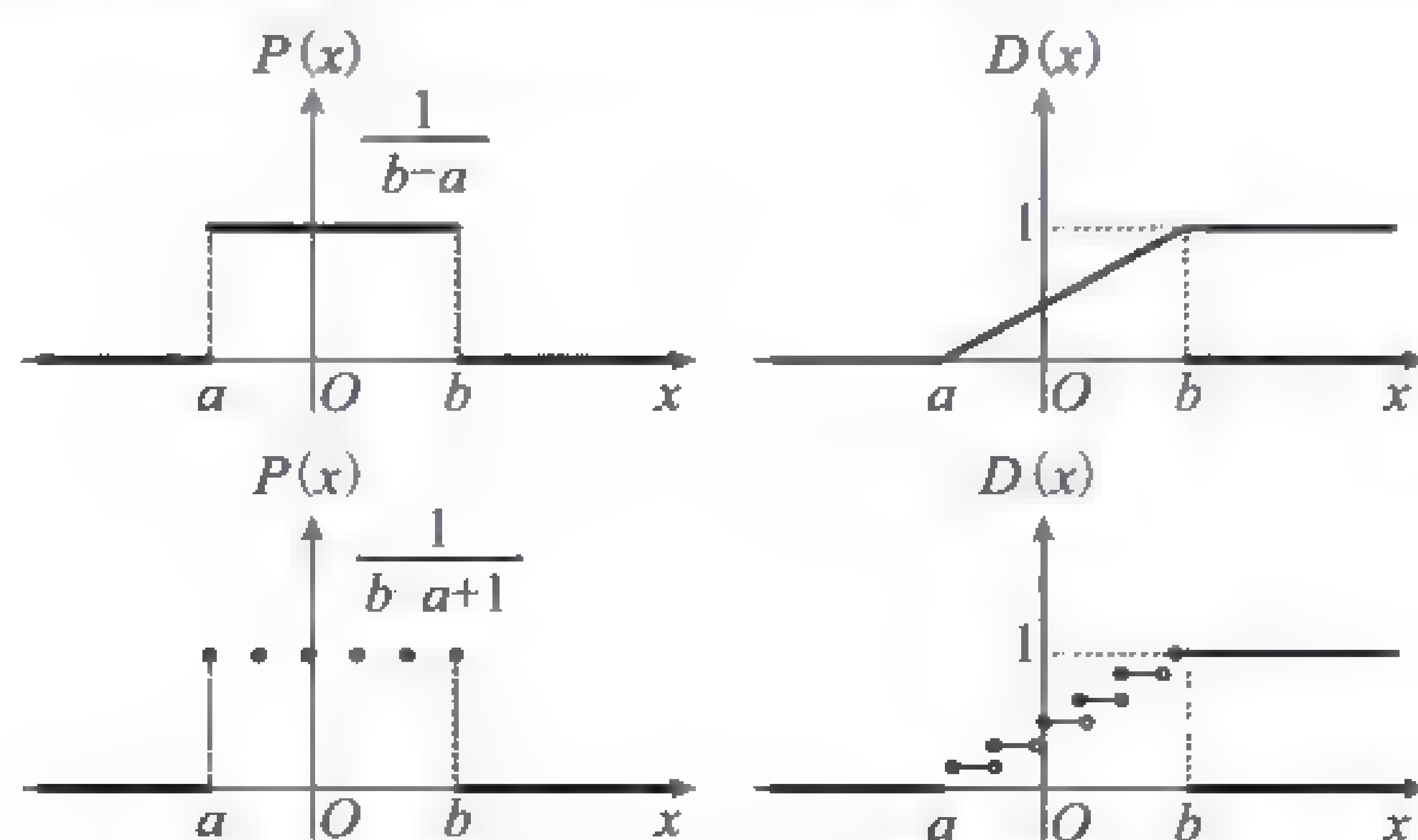


图20-1 均匀分布（上为连续，下为离散）

等概率分布是人们能观察到的最直观的分布特征，比如抛硬币可以得到正反面，分别记为0和1的话，则抛硬币出现正反面和反面的次数就服从离散均匀分布 Unif$\{0, 1\}$。同理，抛一个麻将骰子可以得到6个数字，得到任何一个数字的概率是均等的，则称抛骰子出现的点数服从 Unif$\{1, 2, 3, 4, 5, 6\}$ 的离散均匀分布。连续均匀分布则不然，它在区间上任何一个点上都可能出现，因此连续均匀分布 Unif$(0, 1)$ 则可能出现大于等于0且小于等于1的任何一个小数，我们不可能枚举所有可能出现的情况。

对于每一种分布特征，最重要的是抓住其概率密度函数和累积分布函数特征以及4个最重要的统计量：均值、方差、偏度和峰度。连续均匀分布的概率密度函数和累积分布函数为

$$P(x)=\begin{cases}0, & x<a\\ \dfrac{1}{b-a}, & a\leqslant x\leqslant b\\ 0, & x>b\end{cases}$$

$$D(x)=\begin{cases}0, & x<a\\ \dfrac{x-a}{b-a}, & a\leqslant x\leqslant b\\ 1, & x>b\end{cases}$$

其4个核心统计量：均值μ、方差σ^2、偏度γ_1和峰度为γ_2为

$$\mu=\frac{1}{2}(b+a)$$

$$\sigma^2=\frac{1}{12}(b-a)^2$$

$$\gamma_1=0$$

$$\gamma_2=-\frac{6}{5}$$

对于离散型变量，离散均匀分布的概率密度函数和分布函数如下：

$$P(x)=\begin{cases}0, & x<a\\ \dfrac{1}{b-a+1}, & a\leqslant x\leqslant b\\ 0, & x>b\end{cases}$$

$$D(x)=\begin{cases}0, & x<a\\ \dfrac{x-a+1}{b-a}, & a\leqslant x\leqslant b\\ 1, & x>b\end{cases}$$

其均值和方差为

$$\mu=\frac{1}{2}(b+a)$$

$$\sigma^2=\frac{1}{12}\left((b-a+1)^2-1\right)$$

实际上，离散型均匀分布的总体期望μ和方差σ^2与a和b具体值并没什么关系，而是与变量可能的取值数量N（样本空间）有关。比如抛硬币只有两种可能的情况则$N=2$，抛骰子只有6种可能的情况则$N=6$。因此，对于离散均匀分布的几个核心统计量也可以表示如下：

$$\mu=\frac{1}{2}(N+1)$$

$$\sigma^2=\frac{1}{12}(N-1)(N+1)$$

$$\gamma_1=0$$

$$\gamma_2=-\frac{6\left(N^2+1\right)}{5(N-1)(N+1)}$$

比如扔麻将骰子，总的可能选择 N 为 6（6 个面），得到每个面上的数字为 1,2, …, 6，其中得到每个面的概率都是 1/6，这是一个离散型均匀分布，代入上面的公式可知其总体均值，即数学期望 $\mu=(a+b)/2=(1+6)/2=3.5$，总体方差 $\sigma^2=[(b-a+1)^2-1]/12=35/12=2.9166$。既然离散型均匀分布分布的上下限 a 和 b 与具体值没什么关系，我们也可直接使用样本空间 $N=6$ 进行计算，则总体均值 $\mu=(N+1)/2=(6+1)/2=3.5$，总体方差 $\sigma^2=(N-1)(N+1)/12=5\times7/12=2.9166$。

在 SAS 中，我们可用程序 20-1 生成均匀分布的随机数，然后检查它生成的随机变量服从何种概率分布曲线。从图 20-2 中可以看出在 0~1 之间所有的直方图几乎是等高构成一个矩形分布，其核密度曲线也显示为一个平顶的矩形形状。

程序20-1　连续均匀分布随机数的直方图和核密度估计曲线

```
data sample;
  do i = 1 to 100000;
    x=RAND('UNIFORM') ; /* 0 < x < 1  */
    output;
  end;
  keep x;
run;
proc sgplot data=sample;
  title 'UNIFORM Distribution ';
  histogram x     ;

  density x    / type=normal legendlabel='Normal'
    lineattrs=(pattern=solid color=red);
  density x    / type=kernel legendlabel='UNIFORM'
    lineattrs=(pattern=solid color=blue);
  keylegend / location=inside position=topright across=1;
  xaxis display=(nolabel);
run;

proc means data=sample mean var skewness kurtosis;
run;
```

系统输出为

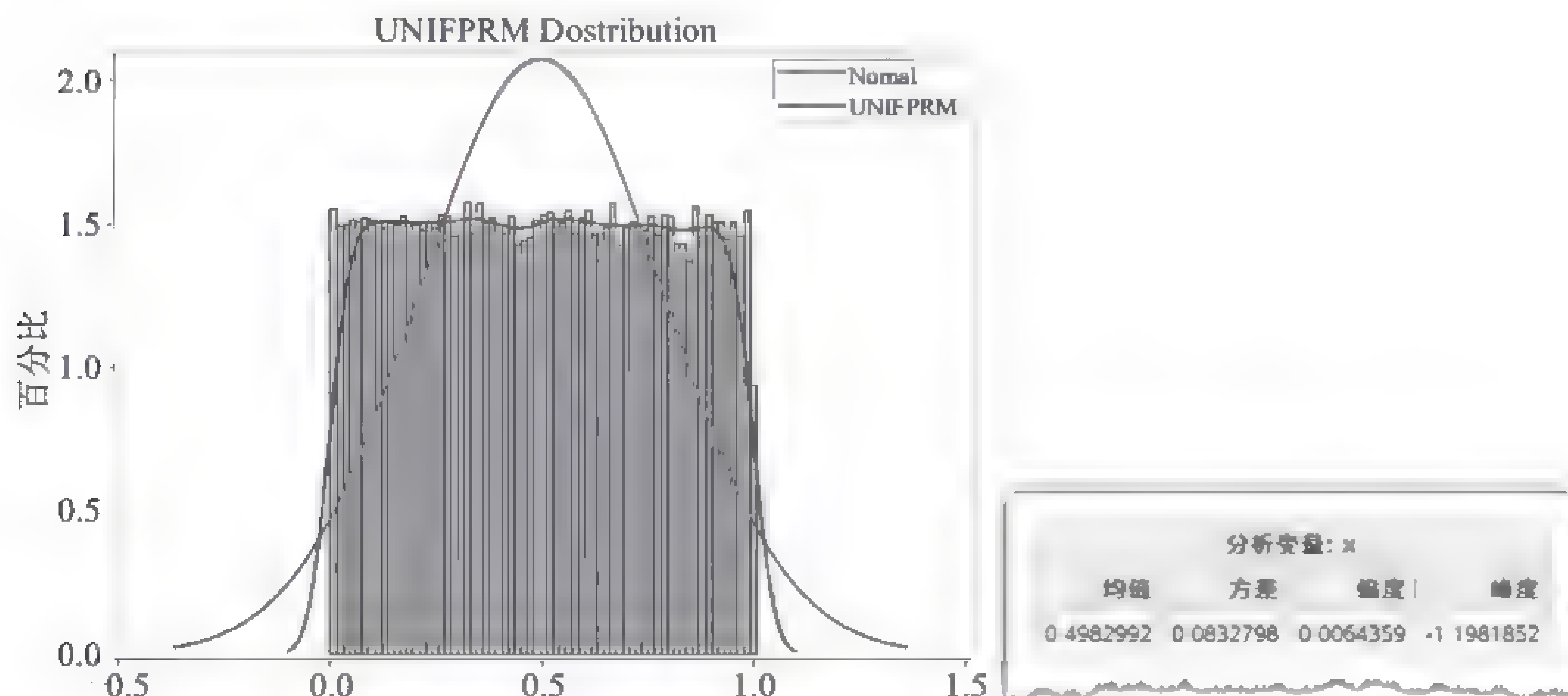


图 20-2　均匀分布的核密度曲线与基本统计量

除了均匀分布支持离散型和连续性变量外，大部分的统计分布分别适用于离散型变量或者连续型变量。因此统计分布可分为离散型分布和连续型分布两大类，下面分别介绍它们。

20.2 离散型统计分布

20.2.1 伯努利分布

只有两种可能结果的单次随机试验称为伯努利试验，比如抛硬币试验。如果定义国徽一面朝上为成功，记为1，则相反的情况国徽朝下定义为试验失败，记为0。伯努利分布（Bernoulli Distribution）是单次试验中只有两种可能结果的离散分布，假定伯努利事件成功发生的概率记为 p，则失败的概率为 $q=1-p$，伯努利分布通常记为 Bern(p)。抛一枚正常硬币国徽朝上的概率服从 $p=0.5$ 的伯努利分布，但如果有人对硬币动过手脚，比如该硬币国徽的反面采用密度更大的铅金属制作而成，则国徽超上的概率会服从 $p>0.5$ 的伯努利分布，如 $p=0.6$。伯努利分布图形如图 20-3 所示。

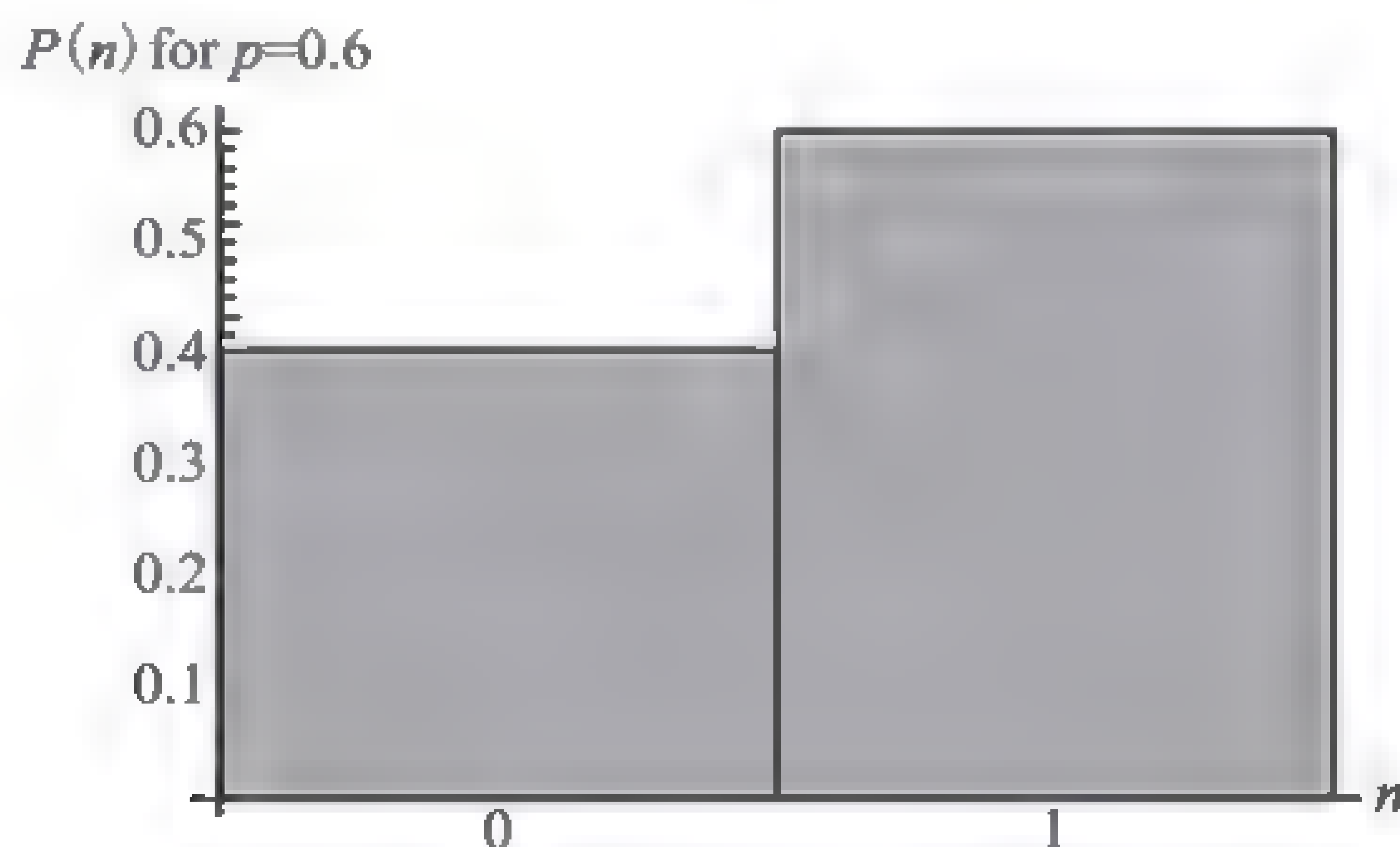


图 20-3　伯努利分布

其中 n 表示试验失败或成功，分别用0和1表示。所以 $p(n=0)$ 和 $p(n=1)$ 分别表示失败和成功的概率。即抛若干次硬币，国徽朝下的次数和朝上的次数与总的抛硬币次数的比。也就是说抛一次硬币就是一个 $p=q=1/2$ 的伯努利试验，国徽朝上和朝下概率分布服从伯努利分布。从这个角度讲，伯努利分布是最简单的离散分布，也是构成其他复杂离散分布的基础。伯努利分布的概率密度函数为

$$P(n)=\begin{cases}1-p, n=0\\ p, n=1\end{cases}$$

为简便起见，伯努利分布的概率密度函数通常记为 $P(n)=p^n(1-p)^{1-n}$，其中 n 可为0或1，分别表示试验失败和试验成功。相应的累积分布函数为

$$D(n)=\begin{cases}1-p, n=0\\ 1, n=1\end{cases}$$

为简便起见，伯努利的累积分布函数通常写作 $D(n)=(1-p)^{1-n}$。

伯努利分布的 4 个核心统计量：均值、方差、偏度和峰度分别为

$$\mu = p$$

$$\sigma^2 = p(1-p)$$

$$\gamma_1 = \frac{1-2p}{\sqrt{p(1-p)}}$$

$$\gamma_2 = \frac{6p^2-6p+1}{p(1-p)}$$

若重复 N 次的独立伯努利试验中，每次试验都只有两种可能的结果，两种结果的发生与否相互独立，与其他各次的试验结果无关，事件发生与否的概率在每一次独立试验中都是固定不变的，则这一系列试验称为 N 重伯努利试验。

设 N 为重复试验次数，n 为重复 N 次试验中成功的次数。定义估计$\hat{p}$为重复 N 次试验中的成功概率为$\hat{p}=\frac{n}{N}$，则 N 次试验中获得 n 次成功的概率为

$$\binom{N}{n}p^n(1-p)^{N-n}$$

其中$\binom{N}{n}$为二项式系数，该系数序列构成了帕斯卡三角式，也称杨辉三角式，如图 20-4 所示。

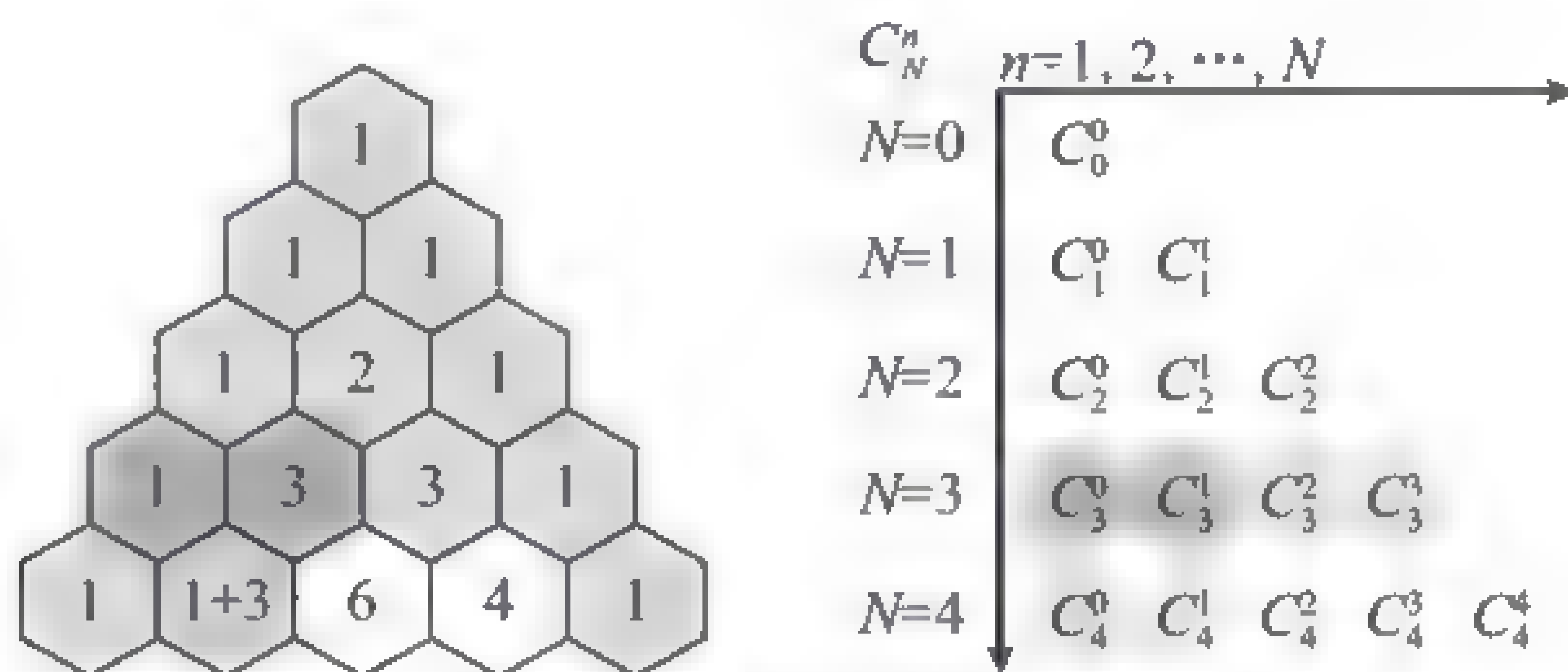


图 20-4　二项式系数构成帕斯卡三角式

则估计$\hat{p}$的期望值可以如下算出：

$$\langle\hat{p}\rangle=\sum_{n=0}^{N}p\binom{N}{n}p^n(1-p)^{N-n}=(1-p)^N\left(\frac{1}{1-p}\right)^N p=p$$

可以看出估计量$\hat{p}$确实是总体均值 p 的无偏估计。N 重伯努利试验中成功次数服从二项分布。从伯努利分布可以衍生出以下各种离散型统计分布：

（1）在 N 次试验中成功的次数，服从二项分布。

（2）第 1 次成功之前的失败次数，服从几何分布。

（3）第 x 次成功之前的失败次数，服从负二项分布。

在 SAS 中，绘制 $X\sim$ Bern(0.6) 直方图如程序 20-2 所示，从直方图的高度可以看出，0 和 1 两种情况分别占 40% 和 60%，总的概率为 1.0（见图 20-5）。

程序20-2　Bern（0.6）的伯努利分布直方图和参考正态分布曲线

```
data sample;
  do N = 1 to 100000;
```

```
    p=0.6;
    x = rand("BERNOULLI", p); /*0 ≤ p ≤ 1 */
    output;
  end;
  keep x;
run;
proc sgplot data=sample;
  title 'BERNOULLI Distribution (p=0.6)';
  histogram x / binwidth=0.5 binstart=0 ;
  density x  / type=normal legendlabel='Normal'
    lineattrs=(pattern=solid color=red);
run;
```

系统输出如图 20-5 所示。

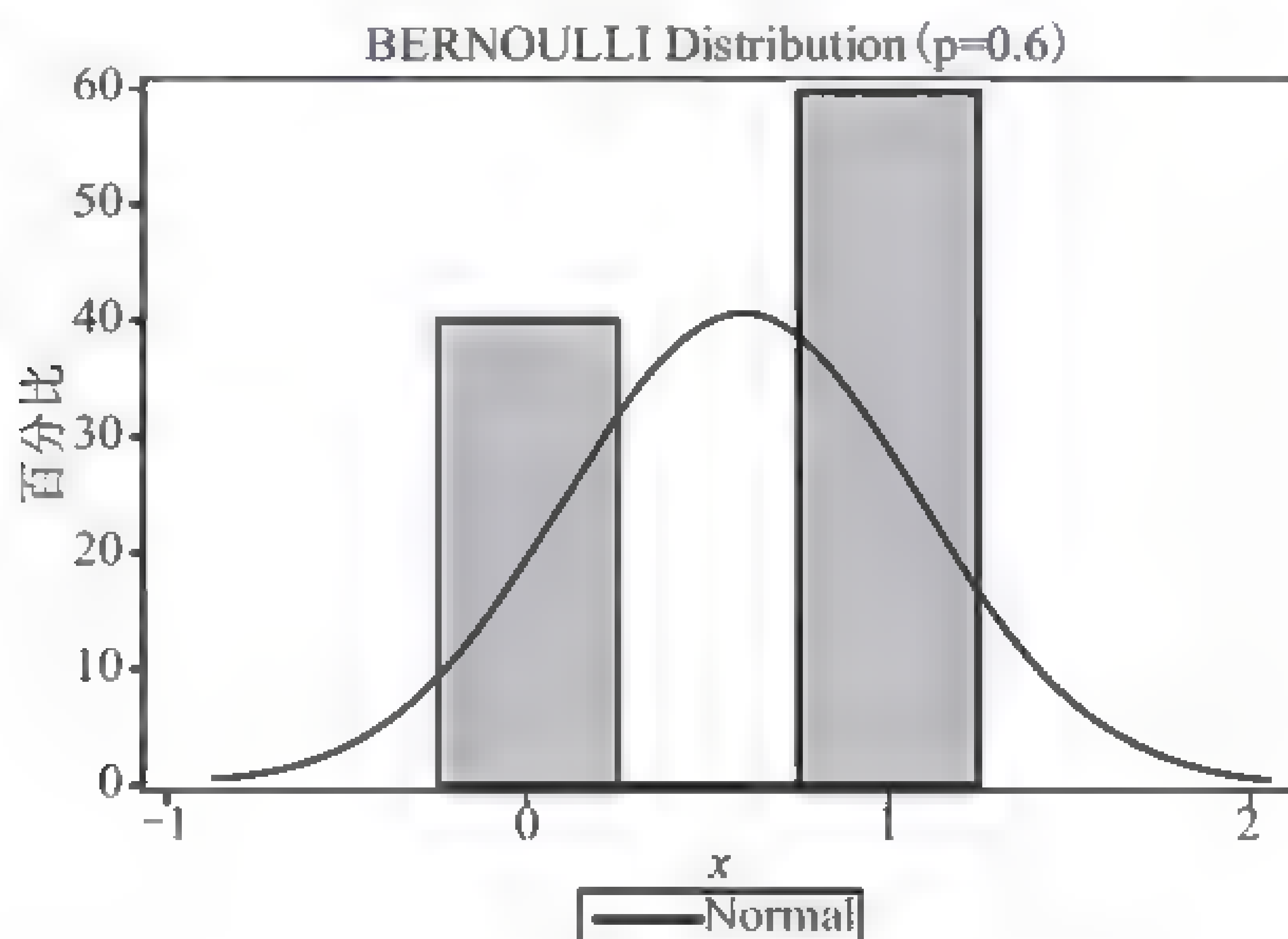


图 20-5　p=0.6 的伯努利分布

我们可以检查该数据的均值、方差、偏度系数和峰度系数，根据公式计算其均值为 0.6（伯努力试验成功次数的期望），方差为 $p\times(1-p)=0.6\times(1-0.6)=0.24$。同时也可用程序 20-3 计算相关统计量进行验证：

```
程序20-3　验证Bern(0.5) 随机数的基本统计量
proc means data=sample mean var skewness;
run;
```

系统输出如图 20-6 所示。

分析变量：x

均值	方差	偏度	峰度
0.6015800	0.2396839	-0.4149803	-1.8278279

图 20-6　p=0.6 的伯努利分布的均值

20.2.2　二项分布

二项分布（Binomial Distribution）是 N 次重复独立的伯努利试验中，获得 n 次成功

的离散概率分布，通常记为 $X\sim \mathrm{Bin}(N,p)$。二项分布简单地说就是把重复 N 次抛硬币当作一次试验，考察其中硬币朝上的次数为 n 的分布情况。二项分布的期望和方差是伯努利分布期望和方差的 N 倍（因为重复 N 次），因此也可以说伯努利分布是二项分布 $N=1$ 的特例。二项分布的概率密度函数为

$$P_p(n\mid N)=\binom{N}{n}p^n q^{N-n}=\frac{N!}{n!(N-n)!}p^n(1-p)^{N-n}$$

其中 $\binom{N}{n}$ 为二项式系数，即 $C_N^n=\dfrac{N!}{n!(N-n)!}$。图 20-7 展示了 $N=20$ 时，抛硬币国徽图案朝上的次数 n 的分布，其中 $p=q=1/2$。

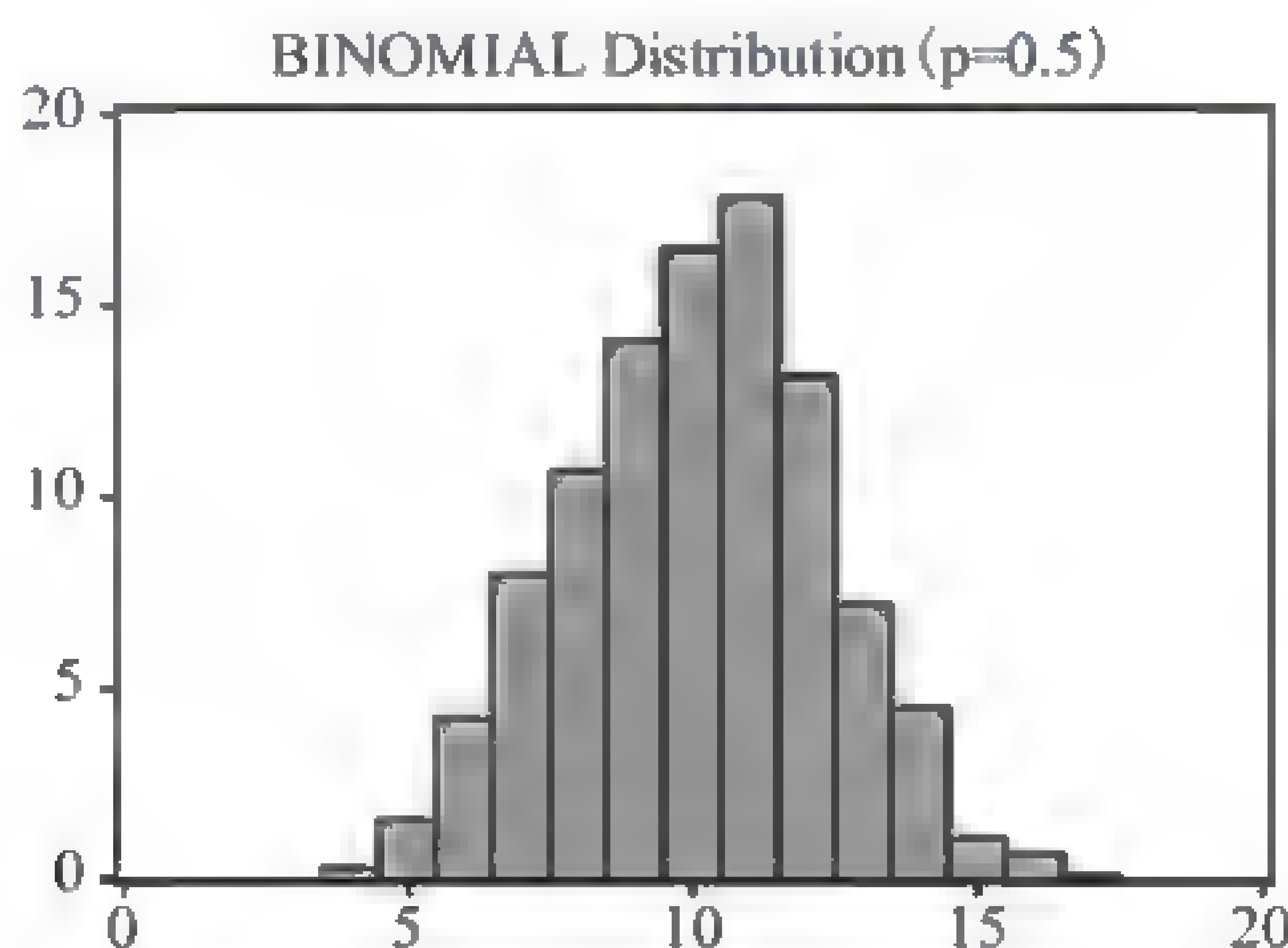


图 20-7　N=20，P=0.5 的二项分布

二项分布的均值、偏度系数和峰度为

$$\mu=N(p+1-p)p=Np$$
$$\sigma^2=Np(1-p)$$
$$\gamma_1=\frac{1-2p}{\sqrt{Np(1-p)}}=\frac{q-p}{\sqrt{Npq}}$$
$$\gamma_2=\frac{6p^2-6p+1}{Np(1-p)}=\frac{1-6pq}{npq}$$

二项分布最典型的例子是重复扔硬币，但它将重复扔 N 次硬币当作是单次试验，其中 X 次正面朝上的概率就是一个二项分布，记为 $\mathrm{Bin}(N,p)$。若我们用 X 代表 N 重伯努利试验中成功的次数，试验成功的概率表示为 p，则 X 的概率分布服从二项分布，记为 $X\sim \mathrm{Bin}(N,p)$。本书附录 1 列出了常用的二项分布累积概率表，用于当 N 不是很大时直接查表。

当试验次数 $N=1$ 时，服从二项分布 $X\sim \mathrm{Bin}(1,p)$，它等价于伯努利分布 $X\sim \mathrm{Bern}(p)$，也等价于区间 [0, 1] 上的离散均匀分布 Unif{0, 1}。

一般地，假定某事件发生的概率为 p，在 N 次试验中，事件发生 x 次的概率就是二项分布。比如假设某种药物治愈率为 0.6，则如果有 100 个人中有 n 个人痊愈的概率为 $Bin(100, 0.6)$，它等于 $C_N^n p^n(1-p)^{N-n}=C_{100}^n 0.6^n(1-0.6)^{100-n}$，其中 $n=1, 2, \cdots, 100$。

- 若X服从伯努利分布，则N个伯努利分布的和（N重伯努利试验的成功次数）服从二项分布，即$X_i \sim \text{Bern}(p) \Rightarrow \sum_{i=1}^{N} X_i \sim \text{Bin}(N, p)$
- 对于二项分布Bin(N, p)，当重复次数N很大时，且p接近于0或1时，二项分布近似于泊松分布$P_0(Np)$。根据泊松定理，二项分布的极限分布为泊松分布，即若当$N \to \infty$时，$Np \to \lambda$，有$C_N^n p^n (1-p)^{N-n} \to \frac{\lambda^n}{n!} e^{-\lambda}$其中$n = 0, 1, 2, \cdots, N$。
- 对于二项分布Bin(N, p)，当重复次数N很大时，而p不接近于0或1，那么可以由棣莫弗-拉普拉斯中心极限定理证明，该二项分布近似于正态分布$X \sim N(Np, Np(1-p))$。棣莫弗-拉普拉斯中心极限定理是中心极限定理的特例，实践中一般当$N \geqslant 30$，$Np \geqslant 10$时逼近正态效果较好。
- 二项分布之和依然服从二项分布，即$X \sim \text{Bin}(N, p)$和$Y \sim \text{Bin}(M, p)$，则有$X+Y \sim \text{Bin}(N+M, p)$。

二项式$(p+q)^N$的展开式将二项分布与二项式系数C_N^n紧密关联起来，而二项式系数正是前面讲到过的帕斯卡三角式中的那些数值。

$$(p+q)^N = \sum_{n=0}^{N} C_N^n p^n q^{N-n} = \sum_{n=0}^{N} \text{Bin}(n, p)$$

SAS 程序 20-4 生成了N=20，p=0.5 的二项分布随机数并绘制直方图，输出结果见前面的图 20-7 所示。

程序20-4　Bin(20, 0.5) 二项分布的直方图（结果见图20-7）

```
data sample;
  do t = 1 to 1000;
    p=0.5;n=20;
      x = rand('BINOMIAL', p, n);
      output;
    end;
  keep x;
run;
proc sgplot data=sample;
  title 'BINOMIAL Distribution (p=0.5)';
  histogram x   / binwidth=1 binstart=1 ;
  xaxis min=1 max=20 integer grid minorgrid  ;
  yaxis min=1 max=20 integer grid minorgrid;
run;
```

在 SAS 语言中，除了使用服从特定统计分布的随机数发生器、创建数据、生成统计分布图外，我们还可以使用概率密度函数 pdf 直接绘制特定统计分布图。程序 20-5 利用 PDF 函数绘制了二项分布的概率密度曲线。

程序20-5　二项分布的概率密度曲线

```
data sample;
  do x=0 to 40  ;
    p=pdf('BINOM', x, 0.3,40);
    p2=pdf('BINOM', x, 0.6,30);
    p3=pdf('BINOM', x, 0.9,25);
    output;
  end;
run;
```

```
proc sgplot;
  title "Binomial";
  series x=x y=p / MARKERS legendlabel="n=40 p=0.3"
    lineattrs=(pattern=solid color=red);
  series x=x y=p2 / MARKERS legendlabel="n=30 p=0.6"
    lineattrs=(pattern=dash color=green);
  series x=x y=p3 / MARKERS legendlabel="n=25 p=0.9"
    lineattrs=(pattern=dot color=blue); ;
  keylegend / location=inside position=topright across=1;
run;
```

系统输出的二项式概率密度分布如图 20-8 所示。

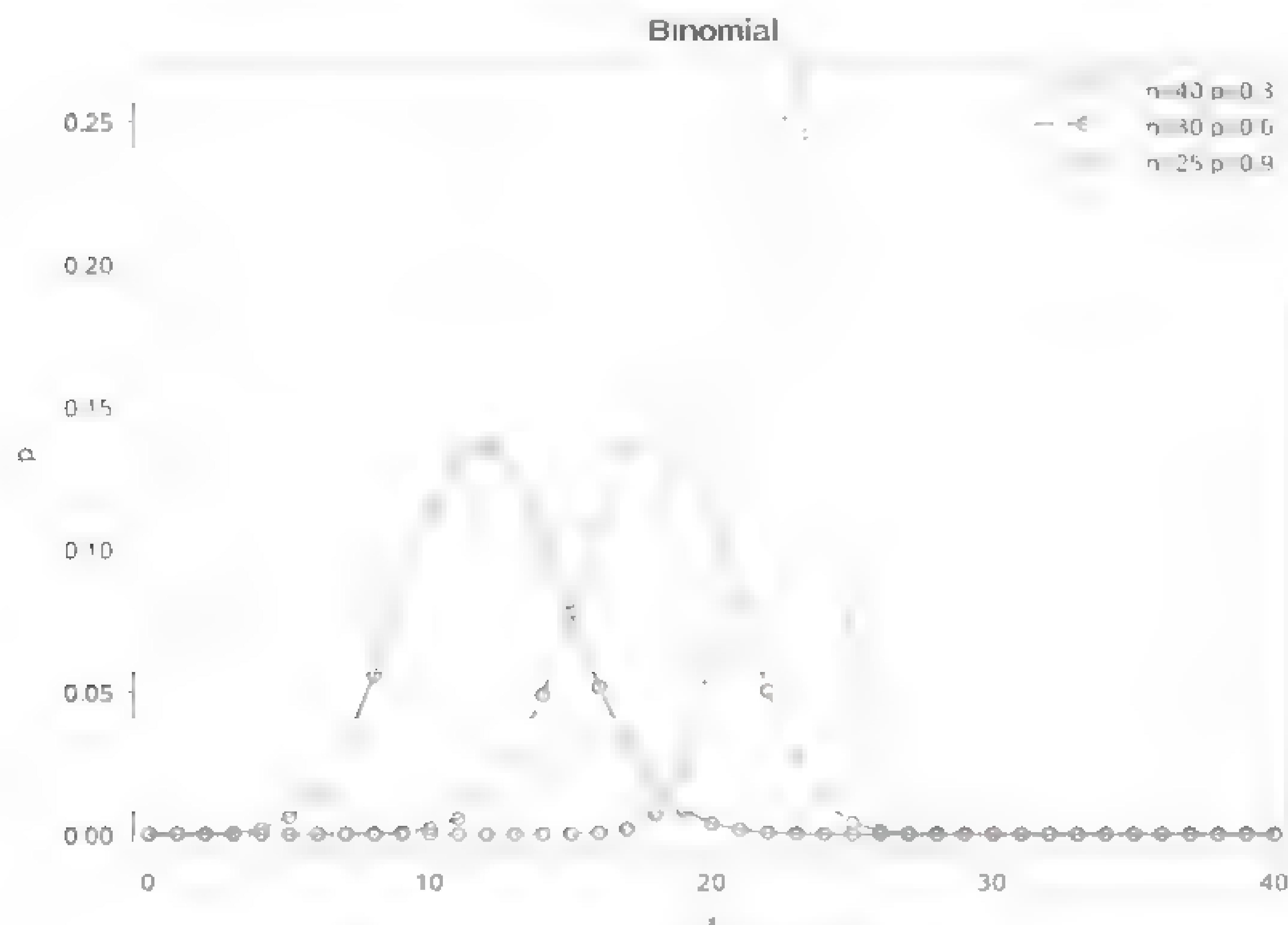


图 20-8　二项式分布的概率密度曲线

同理，我们也可以使用累积分布函数 cdf 绘制对应分布的累积分布图（见程序 20-6）。

程序20-6　二项分布的累积分布曲线

```
data sample;
  do x=0 to 40;
    p=cdf('BINOM', x, 0.3,40);
    p2=cdf('BINOM', x, 0.6,30);
    p3=cdf('BINOM', x, 0.9,25);
    output;
  end;
run;
proc sgplot;
  title "Binomial";
  series x=x y=p / MARKERS legendlabel="n=40 p=0.3"
    lineattrs=(pattern=solid color=red);
  series x=x y=p2 / MARKERS legendlabel="n=30 p=0.6"
    lineattrs=(pattern=dash color=green);
  series x=x y=p3 / MARKERS legendlabel="n=25 p=0.9"
    lineattrs=(pattern=dot color=blue); ;
  keylegend / location=inside position=bottomright across=1;
run;
```

系统输出如图 20-9 所示。

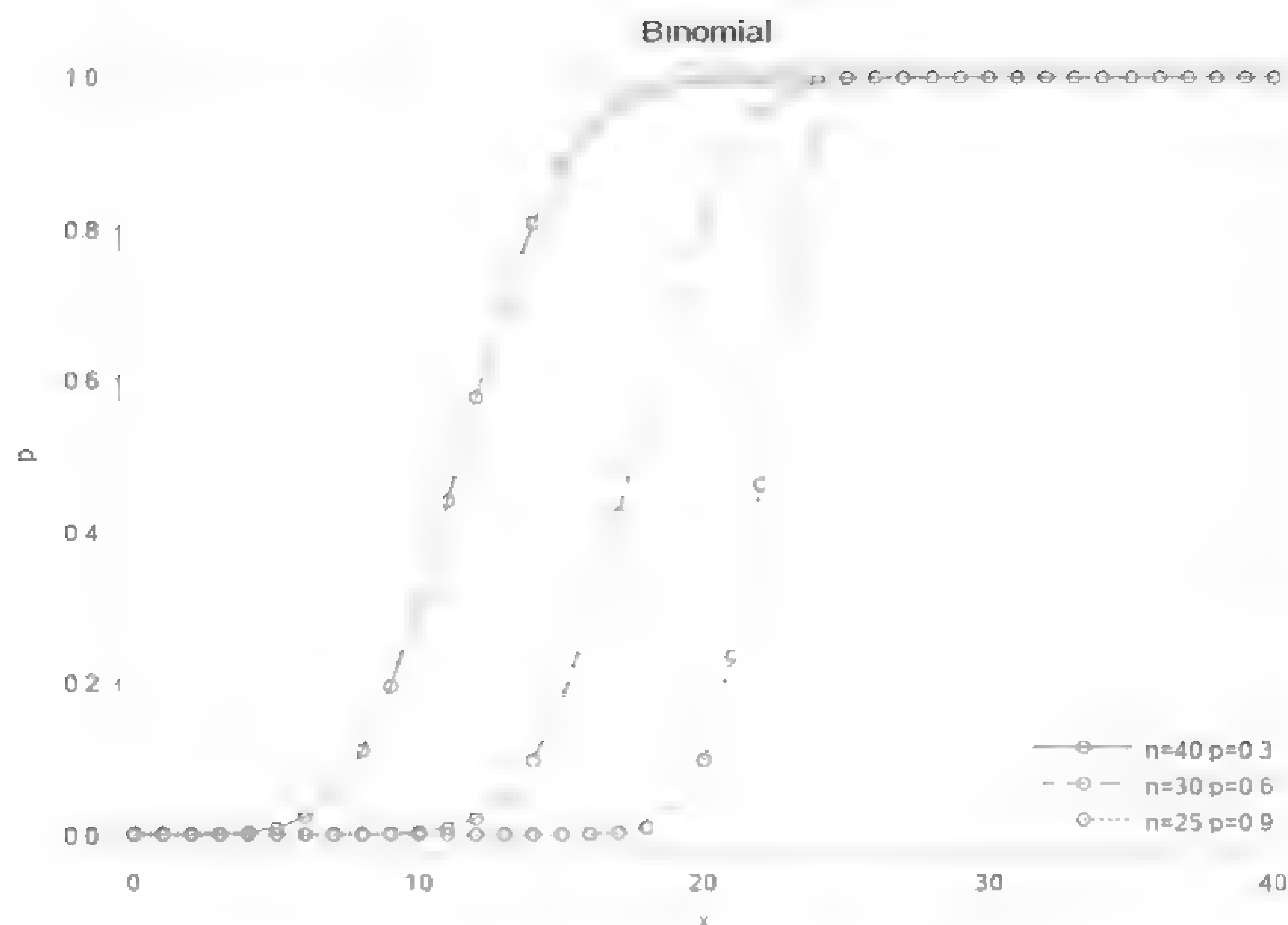


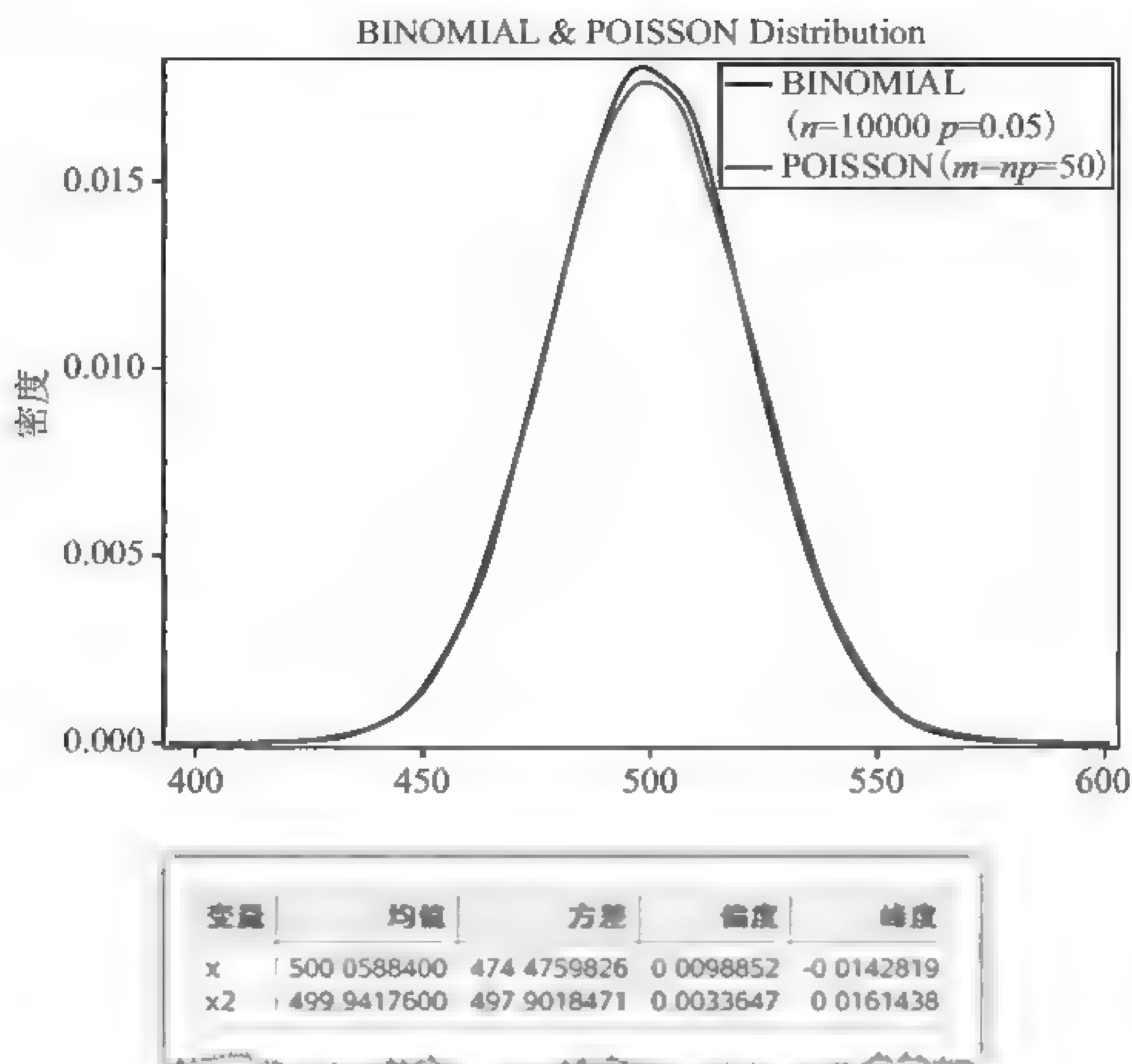
图 20-9 二项式分布的累积分布曲线

根据二项式分布的一个性质，当 n 很大时，并且 p 接近于 0 或者 1，比如 0.95 或者 0.05 时，二项分布 Bin(n, p) 逼近 $\lambda=np$ 的泊松分布 P_o(np)。程序 20-7 及其输出图 20-10 可以验证这一性质。

程序20-7 泊松定理：当很大时，且接近于0或者1，二项分布逼近泊松分布

```
data sample;
  do t = 1 to 100000;
    n=10000; p=0.05;
    x=RAND('BINOMIAL', p, n);

    m=n*p;
    x2=RAND('POISSON', m);
    output;
  end;
  keep x x2  ;
run;
proc sgplot data=sample;
  title 'BINOMIAL & POISSON Distribution  ';
  density x / type=kernel legendlabel='BINOMIAL (n=10000 p=0.05)'
    lineattrs=(pattern=solid color=blue);
  density x2 / type=kernel legendlabel='POISSON (m=np=50)'
    lineattrs=(pattern=solid);
  keylegend / location=inside position=topright across=1;
  xaxis display=(nolabel);
run;
proc means data=sample mean var skewness;
run;
```



变量	均值	方差	偏度	峰度
x	500.0588400	474.4759826	0.0098852	-0.0142819
x2	499.9417600	497.9018471	0.0033647	0.0161438

图 20-10　二项分布与泊松分布

另外，当 n 很大时且 p 不接近于 0 或 1 时，如 p=0.5，二项分布逼近正态分布，此时二项分布的概率密度曲线和正态分布曲线拟合极好。其对应 SAS 验证代码如程序 20-8 所示。

程序20-8　棣莫弗-拉普拉斯CLT：当n很大且P不接近于0或1时，二项分布逼近正态分布

```
data sample;
  do t = 1 to 100000;
    n=10000; p=0.5;
    x=RAND('BINOMIAL', p, n); /*n=1,2,...*/
    output;
  end;
  keep x;
run;
proc sgplot data=sample;
  title 'BINOMIAL Distribution (p=0.5 n=20)';
  histogram x ;
  density x / type=normal legendlabel='Normal'
    lineattrs=(pattern=solid color=red);
  density x / type=kernel legendlabel='BINOMIAL (n=10000
    p=0.5)' lineattrs=(pattern=solid);
  keylegend / location=inside position=topright across=1;
  xaxis display=(nolabel);
run;
proc means data sample mean var skewness kurtosis; run;
```

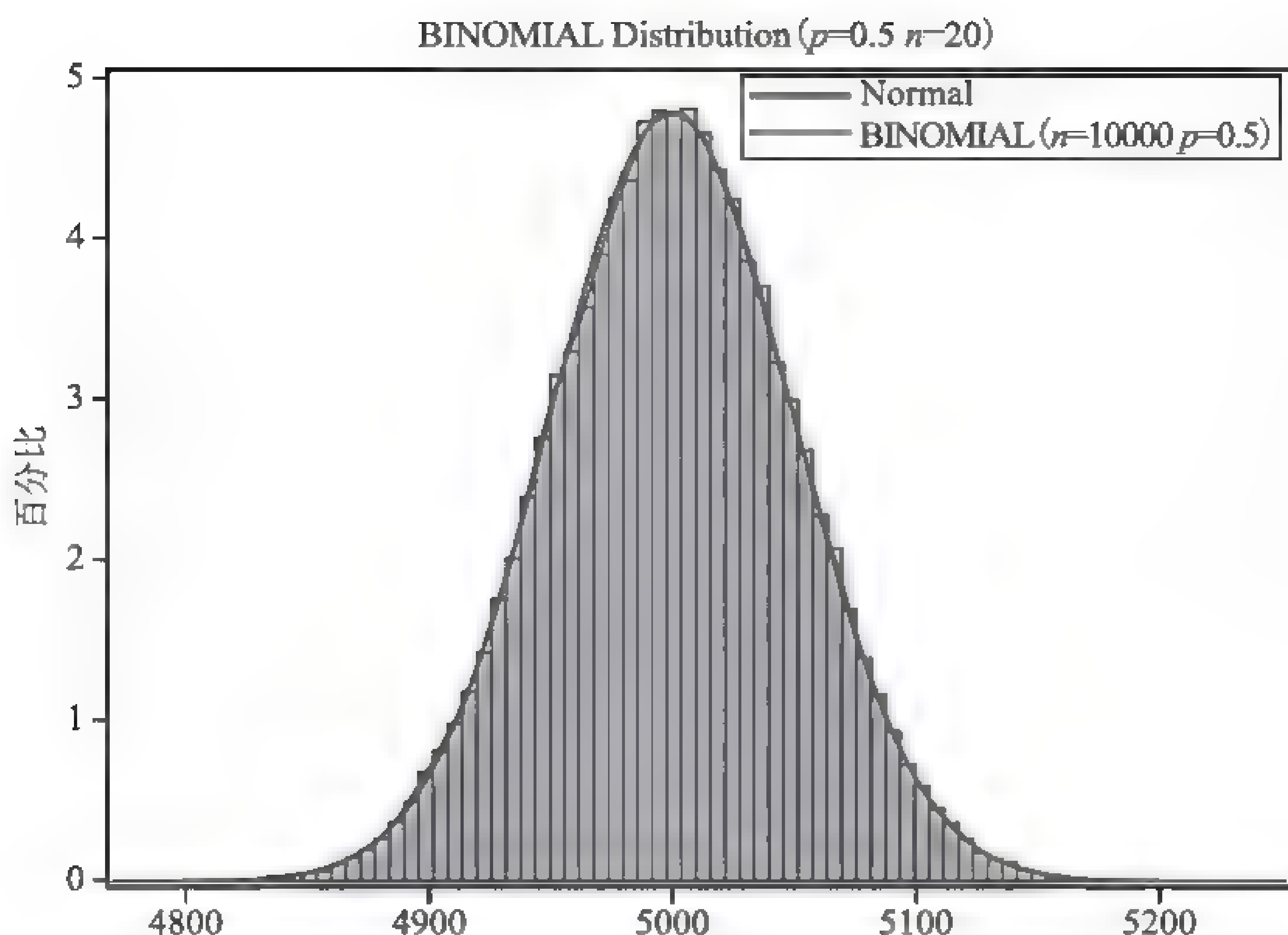


图 20-11　二项分布与正态分布

20.2.3　几何分布

对于大于等于零的整数离散变量 n 其中 n=0, 1、2、…，服从如下概率密度函数 $P(n)$，则称 n 服从几何分布（Geometric Distribution），通常记为 Geo(p) 或 G(p)，其中 $0 < p < 1$ 且 q=1−p。

$$P(n) = p(1-p)^n = pq^n$$

$$D(n) = \sum_{k=0}^{n} p(k) = 1 - q^{n+1}$$

在 N 重伯努利试验中获得试验成功之前的失败次数所服从的分布就是几何分布，比如我们不停地扔硬币，直到我们得到国徽朝上（试验成功）的结果，则前面的试验次数之和服从几何分布。根据上面几何分布定义公式，所有的概率值之和为 1。

$$\sum_{n=0}^{\infty} p(n) = \sum_{n=0}^{\infty} pq^n = p\sum_{n=0}^{\infty} q^n = \frac{p}{(1-q)} = \frac{p}{p} - 1$$

图 20-12 为 p=0.5 的几何分布示例。

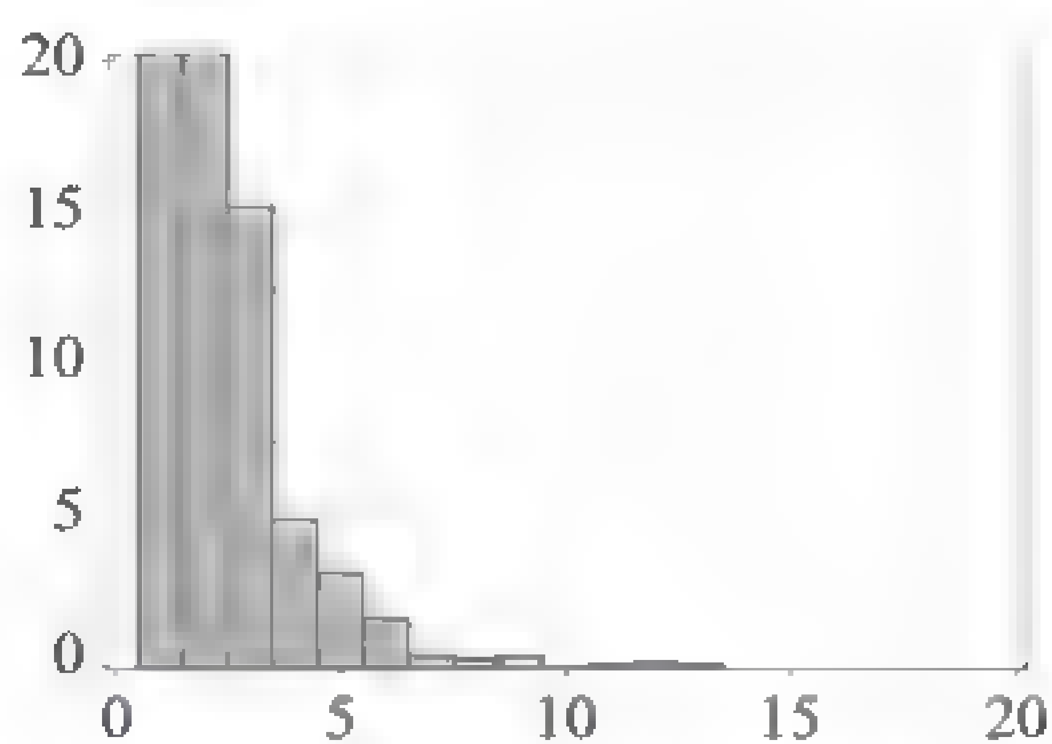


图 20-12　p=0.5 的几何分布

几何分布的总体均值、方差、偏度和峰度为

$$\mu=\frac{1-p}{p}$$

$$\sigma^2=\frac{1-p}{p^2}$$

$$\gamma_1=\frac{2-p}{\sqrt{1-p}}$$

$$\gamma_2=\frac{p^2-6p+6}{1-p}$$

在离散型统计分布中，几何分布是唯一的离散型无记忆随机分布，是指数分布的离散模拟。而指数分布则是唯一的连续型无记忆分布，是几何分布的连续模拟。几何分布是 N 重伯努利试验中第一次试验成功之前的失败次数，或者获得一次成功所需要的试验次数（两者相差 1）所服从的统计分布。

比如我们定义在投骰子试验中，一直投骰子直到得到点数为 6 的投掷次数，则该投掷次数服从 p=1/6 的几何分布。如果在不断扔硬币的试验中，则是我们得到国徽朝上时之前的投掷次数。因此，均匀分布和伯努利分布都可以导出几何分布。下面为 SAS 绘制的几何分布的概率密度函数（见程序 20-9 及其输出结果图 20-13）和累积分布函数图（见程序 20-10 及其输出结果图 20-14）。

程序20-9　几何分布的概率密度曲线

```
data sample;
  do x=0 to 10  ;
    p=pdf('GEOMETRIC', x, 0.2);
    p2=pdf('GEOMETRIC', x, 0.5);
    p3=pdf('GEOMETRIC', x, 0.8);
    output;
  end;
run;
proc sgplot;
  title"GEOMETRIC";
  series x=x y=p / MARKERS legendlabel="p=0.2"
    lineattrs=(pattern=solid color=red);
  series x=x y=p2 / MARKERS legendlabel="p=0.5"
    lineattrs=(pattern=dash color=green);
  series x=x y=p3 / MARKERS legendlabel="p=0.8"
    lineattrs=(pattern=dot color=blue); ;
  keylegend / location=inside position=topright across=1;
run;
```

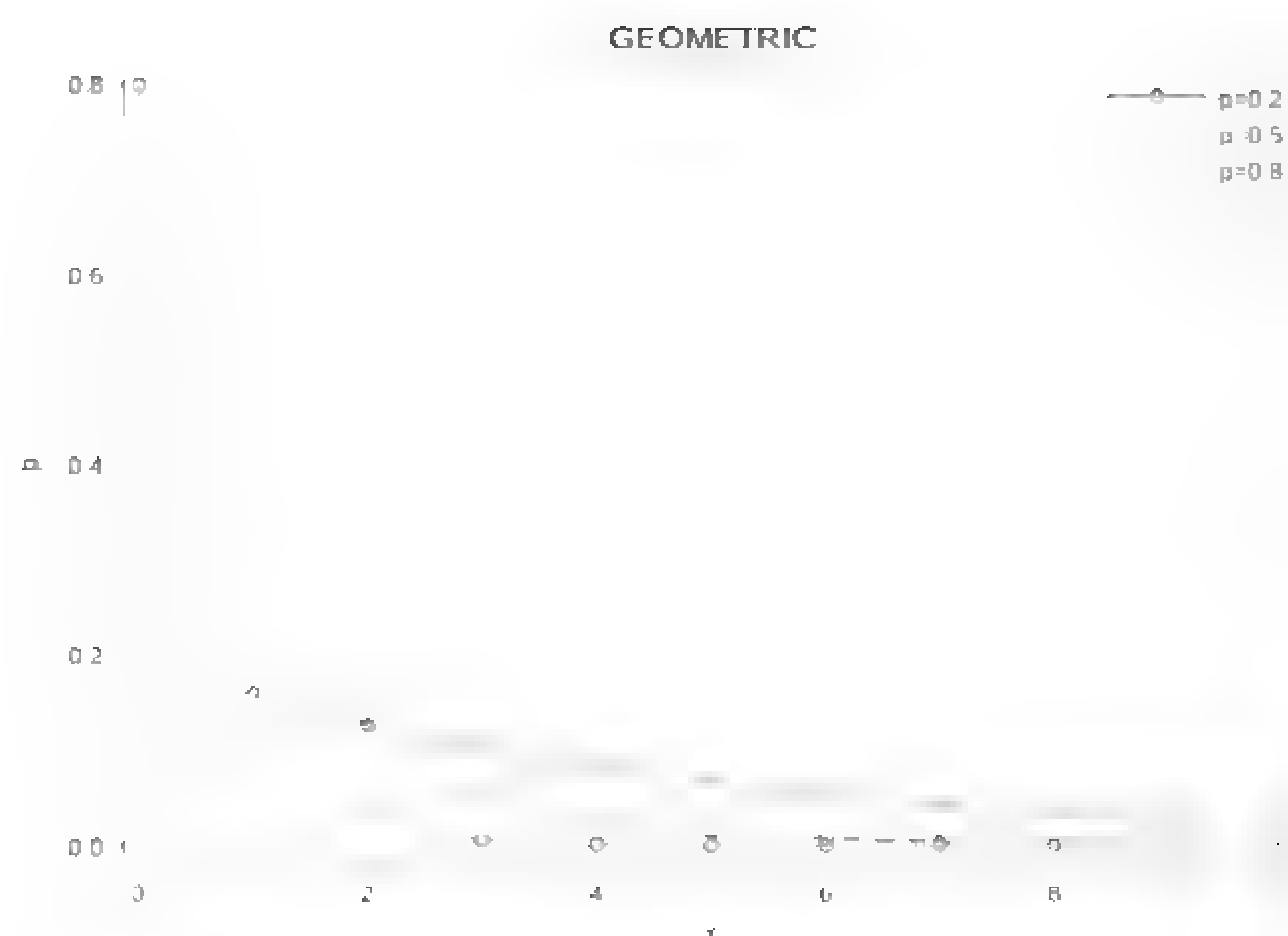


图 20-13　几何分布的概率密度曲线

程序20-10　几何分布的累积分布曲线

```
data sample;
  do x=0 to 10  ;
    p=cdf('GEOMETRIC', x, 0.2);
    p2=cdf('GEOMETRIC', x, 0.5);
    p3=cdf('GEOMETRIC', x, 0.8);
    output;
  end;
run;
proc sgplot;
  title "GEOMETRIC";
  series x=x y=p / MARKERS legendlabel="p=0.2"
    lineattrs=(pattern=solid color=red);
  series x=x y=p2 / MARKERS legendlabel="p=0.5"
    lineattrs=(pattern=dash color=green);
  series x=x y=p3 / MARKERS legendlabel="p=0.8"
    lineattrs=(pattern=dot color=blue); ;
  keylegend / location=inside position=bottomright across=1;
run;
```

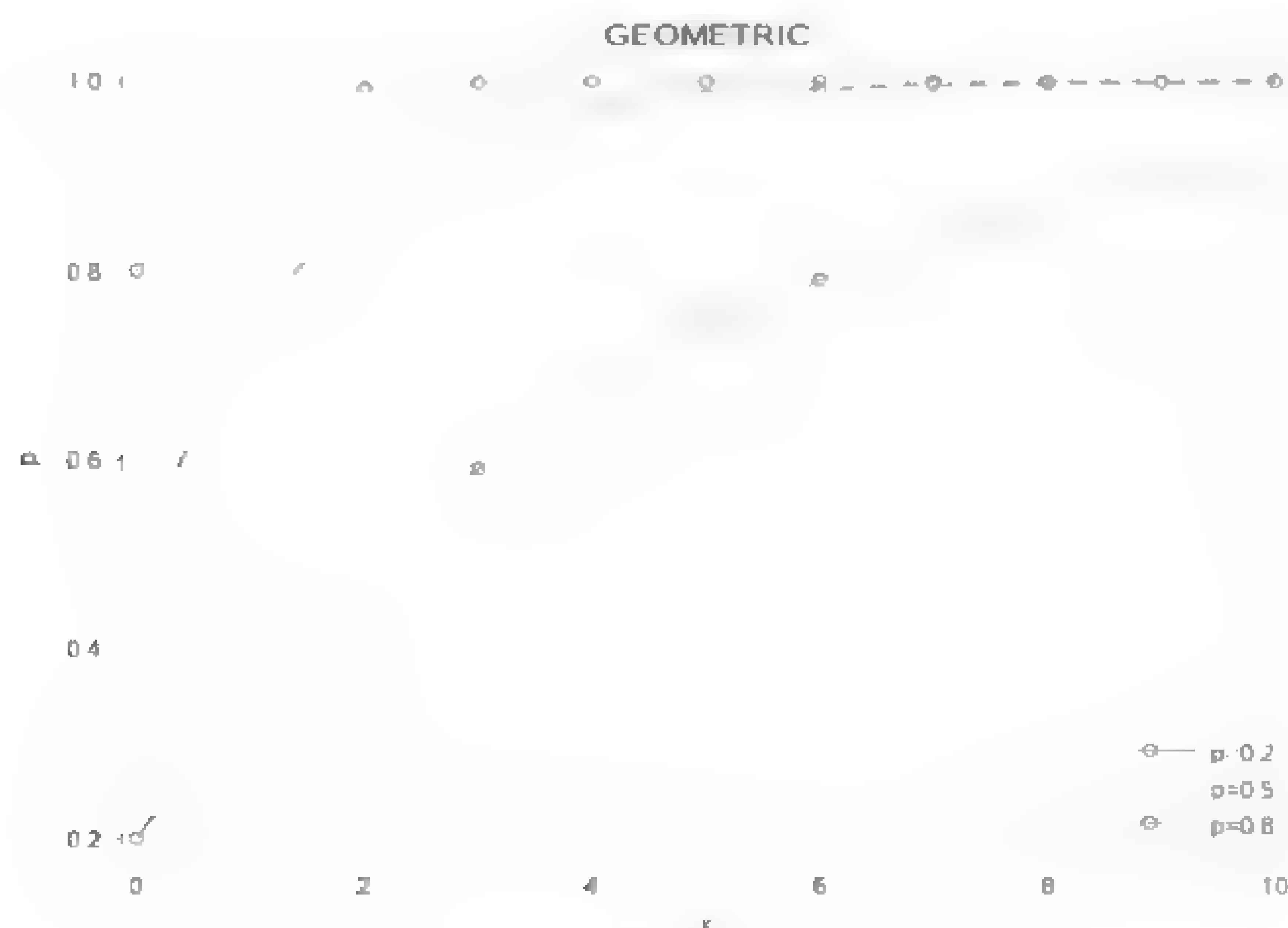


图 20-14　几何分布的累积分布曲线

程序 20-11 展示了如何从均匀分布的随机变量中构建出几何分布，它充分说明各种统计分布之间在某些条件下相互转化的数学规律（输出如图 20-15 所示）。

程序20-11　从均匀分布构造出几何分布：抛骰子抛到6 之前的失败次数

```
data sample;
  n=100000;

  t=0; /*第一次成功之前失败的次数*/
  do i = 1 to n;
    x=floor(RAND('UNIFORM')*6)+1 ; /* 1,2,3,4,5,6 */
    if x=6 then do;
      x=t;
      t=0;
      output;
    end;
    t=t+1;
  end;
  keep x;
run;
```

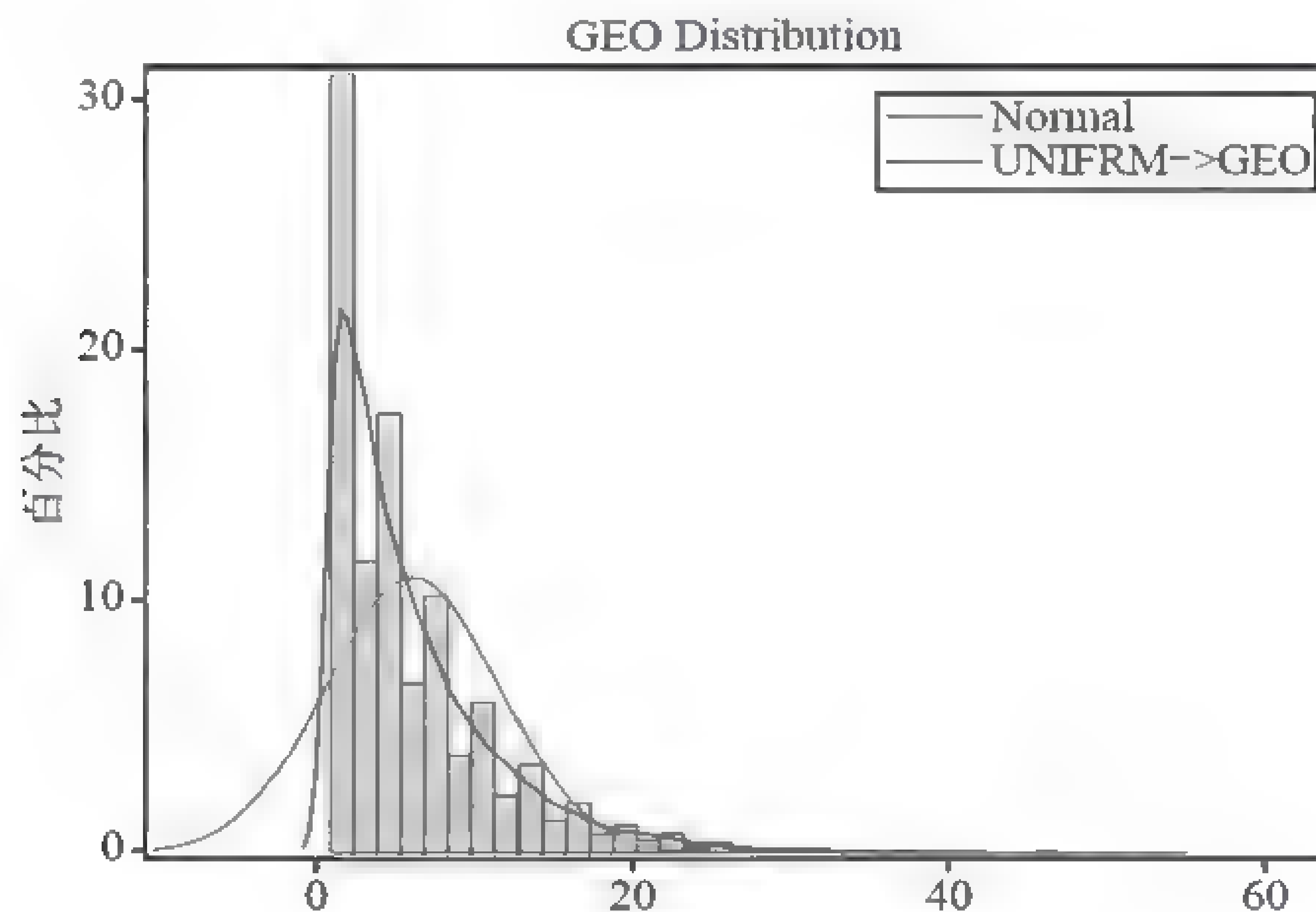


图 20-15　均匀分布与几何分布

程序 20-12 则展示了如何从伯努利分布构造出服从几何分布的随机变量，输出如图 20-16 所示。

程序20-12　从伯努利分布构造出几何分布：抛硬币得到国徽朝上之前的抛币次数

```
data sample;
  n=100000;

  t=0; /*第一次成功之前失败的次数*/
  do i = 1 to n;
    x=RAND('BERNOULLI', 0.5); /* 0, 1*/
    if x=1 then do;
      x=t;/*记录得到朝上之前尝试的次数*/
      t=0;
      output;
    end;
    t=t+1;
  end;
  keep x;
run;
```

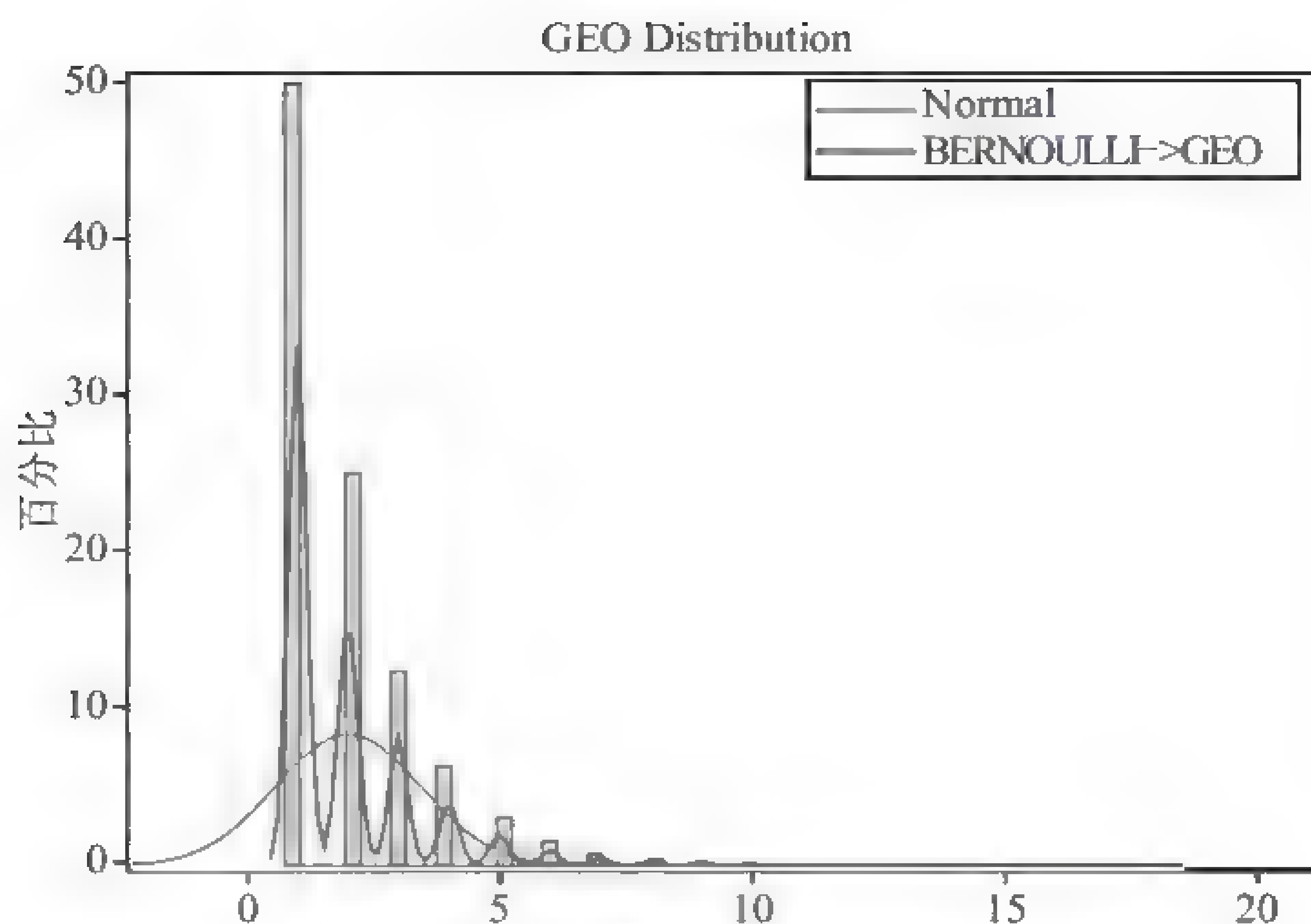


图 20-16　伯努利分布与几何分布

20.2.4　负二项分布

负二项分布（NEGBINOMIAL Distribution）又称帕斯卡分布，它给出多重伯努利试验中第 $x+r$ 次试验成功之前失败次数 x 的概率分布，通常记为 $X\sim N\mathrm{Bin}(r,p)$ 或 $X\sim NB(r,p)$。其中共试验 $x+r-1$ 次，$r-1$ 次成功。其概率密度函数为

$p_{r,p}(x)=\binom{x+r-1}{r-1}p^r(1-p)^x$ 其中 $\binom{x+r-1}{r-1}$ 为二项式系数。

对应的累积分布函数为

$$D(x)=I(p;r,x+1)$$

其中等号右侧为归一化贝塔函数。

负二项分布的总体均值，方差，偏度系数和峰度系数为

$$\mu=\frac{rq}{p}$$

$$\sigma^2=\frac{rq}{p^2}$$

$$\gamma_1=\frac{2-p}{\sqrt{rq}}$$

$$\gamma_2=\frac{p^2-6p+6}{rq}$$

在日常生活中我们要投掷骰子，每个面都是 1/6。如果我们把投出 3 次相同的点数作为单个试验成功事件，则该试验事件成功之前投掷失败次数就服从负二项分布。如果使用抛硬币，则得到 3 次国徽朝上之前的失败次数也是服从负二项分布。SAS 程序 20-13 展示了伯努利试验导出负二项分布随机变量的过程。

程序20-13　伯努利试验导出负二项分布的过程

```
data sample;
  n=100000;

  t=0; /*投掷次数*/
  s=0; /*成功次数 */
  r=3; /*要求3次成功*/
  do i = 1 to n;
    x=RAND('BERNOULLI', 0.5);
    t=t+1;

    if x=1 then do;
      s=s+1;/*国徽朝上，成功次数加1*/
    end;

    if s=r then do;
      x=t-r;/*记录得到 r 次成功前尝试的次数*/
      t=0;
      s=0;
      output;
    end;
  end;
  keep x;
run;
```

负二项分布具有如下性质：

（1）当 r=1 时，负二项分布退化为几何分布：X~NBin=Geo(p)。

（2）当 $r>1$ 时，负二项分布跟几何分布有倍数关系：$X\sim N\mathrm{Bin}(r,p)=\sum_{i=1}^{r}\mathrm{Geo}(p)$。

（3）任意 k 个负二项分布的随机变量 X_i 之和 $\sum X_i$ 服从 $n=\sum r_i$ 的负二项分布：$\sum X_i\sim N\mathrm{Bin}(\sum r_i,p)$。

SAS 程序 20-14 和程序 20-15 绘制负二项分布的概率密度曲线和累积分布曲线（见图 20-17 和图 20-18）：

程序20-14　负二项分布的概率密度曲线

```
data sample;
  do x=0 to 80 ;
    p=pdf('NEGBINOMIAL', x, 0.25,10);
    p2=pdf('NEGBINOMIAL', x, 0.50,10);
    p3=pdf('NEGBINOMIAL', x, 0.25,20);
    output;
  end;
run;
proc sgplot;
  title "NEGBINOMIAL";
  series x=x y=p / MARKERS legendlabel="r=10 p=0.25"
    lineattrs=(pattern=solid color=red);
  series x=x y=p2 / MARKERS legendlabel="r=10 p=0.50"
    lineattrs=(pattern=solid color=green);
  series x=x y=p3 / MARKERS legendlabel="r=20 p=0.25"
    lineattrs=(pattern=solid color=blue); ;
  keylegend / location=inside position=topright across=1;
run;
```

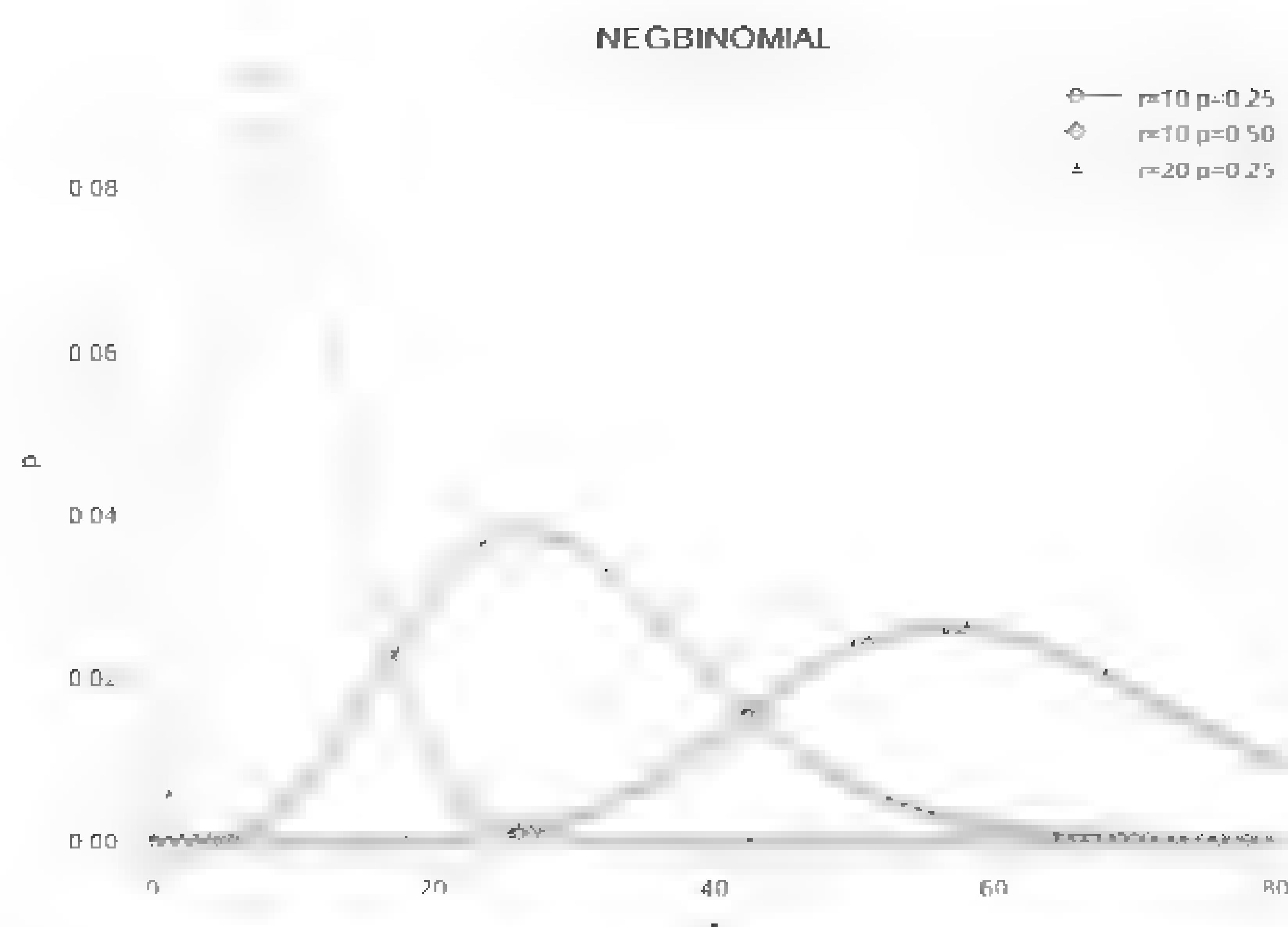


图 20-17　负二项分布的概率密度曲线

程序20-15　负二项分布的累积分布曲线

```
data sample;
  do x=0 to 80  ;
    p=cdf('NEGBINOMIAL', x, 0.25,10);
    p2=cdf('NEGBINOMIAL', x, 0.50,10);
    p3=cdf('NEGBINOMIAL', x, 0.25,20);
    output;
  end;
run;
proc sgplot;
  title "NEGBINOMIAL";
  series x=x y=p / MARKERS legendlabel="r=10 p=0.25"
    lineattrs=(pattern=solid color=red);
  series x=x y=p2 / MARKERS legendlabel="r=10 p=0.50"
    lineattrs=(pattern=solid color=green);
  series x=x y=p3 / MARKERS legendlabel="r=20 p=0.25"
    lineattrs=(pattern=solid color=blue); ;
  keylegend / location=inside position=bottomright across=1;
run;
```

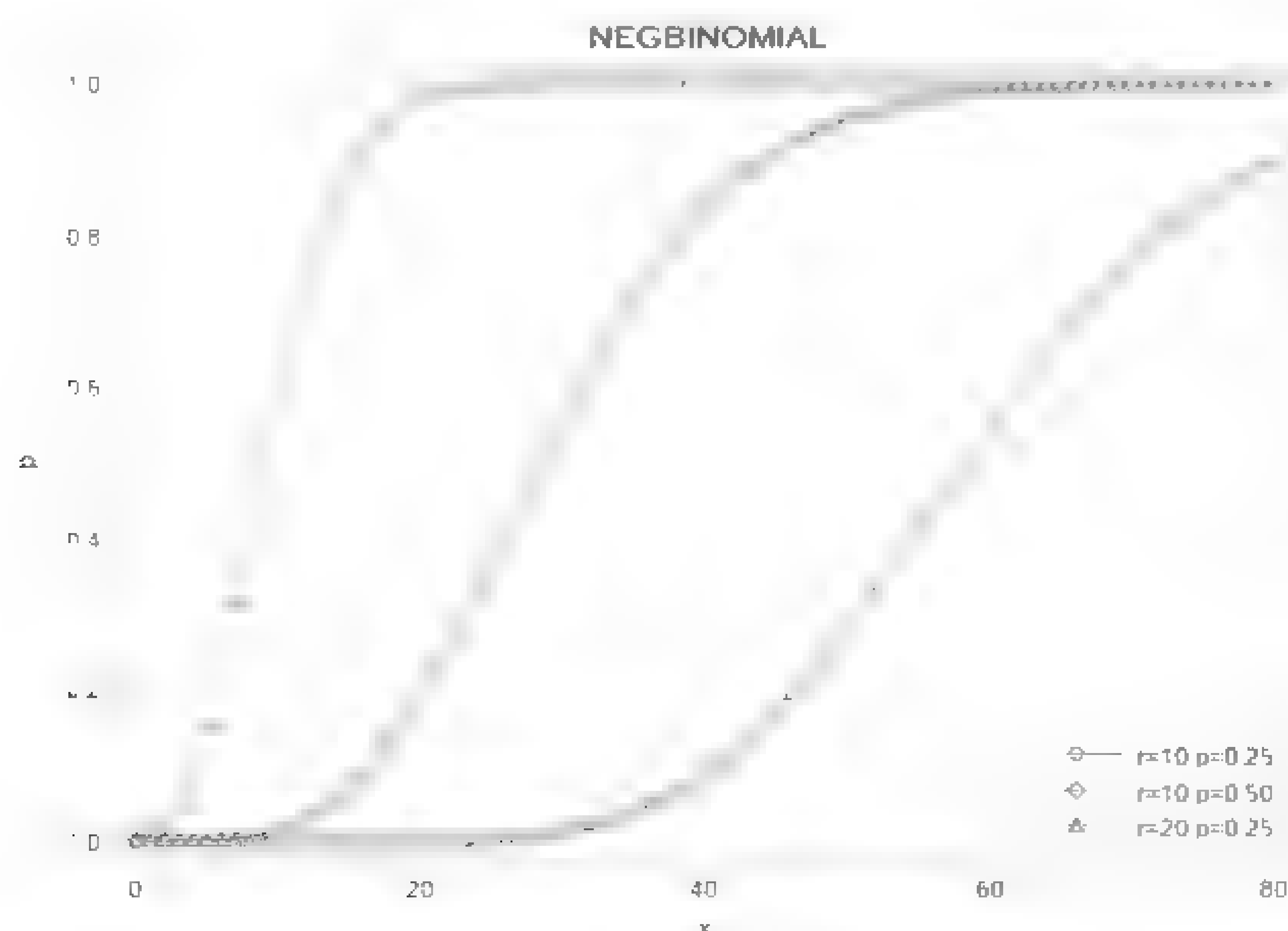


图 20-18　负二项分布的累积分布曲线

20.2.5 超几何分布

假定 $n+m$ 个可能的选择中有 n 个好的选择和 m 个坏的选择，如果选择正确的话记为 1，否则记为 0，即假定第 i 个选择正确的话记 $x_i=1$，否则 $x_i=0$。则成功选择的总数为 $x=\sum_{i=1}^{N}x_i$，第 i 个选择成功的概率记为

$$p(x=i)=\frac{m!n!N!(m+n-N)!}{i!(n-i)!(m+i-N)!(N-i)!(m+n)!}$$

此类问题与从一个包含 n 个白球 m 个黑球的袋子中随机抽取 N 个球，其中有 i 个白球的概率等价。也可以想象从总体为 N 的产品中有 m 件不合格的产品，其不合格率为 $p=m/N$。现在从产品中不放回地随机抽取 n 件产品，其中有 k 个产品不合格，则 k 个产品不合格的概率为

$$P(X=k)=\frac{C_m^k C_{N-m}^{n-k}}{C_N^n}$$

则随机变量 X 所服从的分布称为超几何分布（Hypergeometric Distribution），记为 $X\sim Hyp(N, m, n)$。二项定理的内容就是断言超几何分布的极限分布为二项分布，即当 $N\to\infty$时，$m/N\to p$，有

$$P(X=k)=\frac{C_m^k C_{N-m}^{n-k}}{C_N^n}\to C_n^k p^k(1-p)^{n-k}$$

超几何分布的数学期望、方差和偏度计算公式如下，而峰度为一个包含超几何函数的复杂表达式，此处不再列出。

$$\mu=\frac{nN}{m+n}$$

$$\sigma^2=\frac{mnN(m+n-N)}{(m+n)^2(m+n-1)}$$

$$\gamma_1=\frac{(m-n)(m+n-2N)}{m+n-2}\sqrt{\frac{m+n-1}{mnN(m+n-N)}}$$

超几何分布有 3 个参数，通常记为 $Hyp(N, m, n)$。SAS 程序 20-16 和程序 20-17 绘制出对应的概率密度曲线和累积分布曲线（见图 20-19 和图 20-20）。

程序20-16 超几何分布的概率密度曲线

```
data sample;
  do x=0 to 3;
    N=10;R=3;nn=2;
    p=pdf('HYPER', x, 10,3 ,2 );
    p2=pdf('HYPER', x, 50,3,3);
    p3=pdf('HYPER', x, 10,6,2);
    output;
  end;
run;
proc sgplot;
  title "HYPER";
  series x=x y=p / MARKERS legendlabel="N=10 m=3 n=2"
    lineattrs=(pattern=solid color=red);
  series x=x y=p2 / MARKERS legendlabel="N=50 m=3 n=3"
    lineattrs=(pattern=dash color=green);
  series x=x y=p3 / MARKERS legendlabel="N=10 m=6 n=2"
```

```
    lineattrs=(pattern=dot color=blue);
  keylegend / location=inside position=topright across=1;
run;
```



图 20-19 超几何分布的概率密度曲线

程序20-17 超几何分布的累积分布曲线

```
data sample;
  do x=0 to 3;
    p=cdf('HYPER', x, 10,3 ,2 );
    p2=cdf('HYPER', x, 50,3,3);
    p3=cdf('HYPER', x, 10,6,2);
    output;
  end;
run;
proc sgplot;
  title "HYPER";
  series x=x y=p / MARKERS legendlabel="N=10 m=3 n=2"
    lineattrs=(pattern=solid color=red);
  series x=x y=p2 / MARKERS legendlabel="N=50 m=3 n=3"
    lineattrs=(pattern=dash color=green);
  series x=x y=p3 / MARKERS legendlabel="N=10 m=6 n=2"
    lineattrs=(pattern=dot color=blue);
  keylegend / location=inside position=bottomright across=1;
run;
```

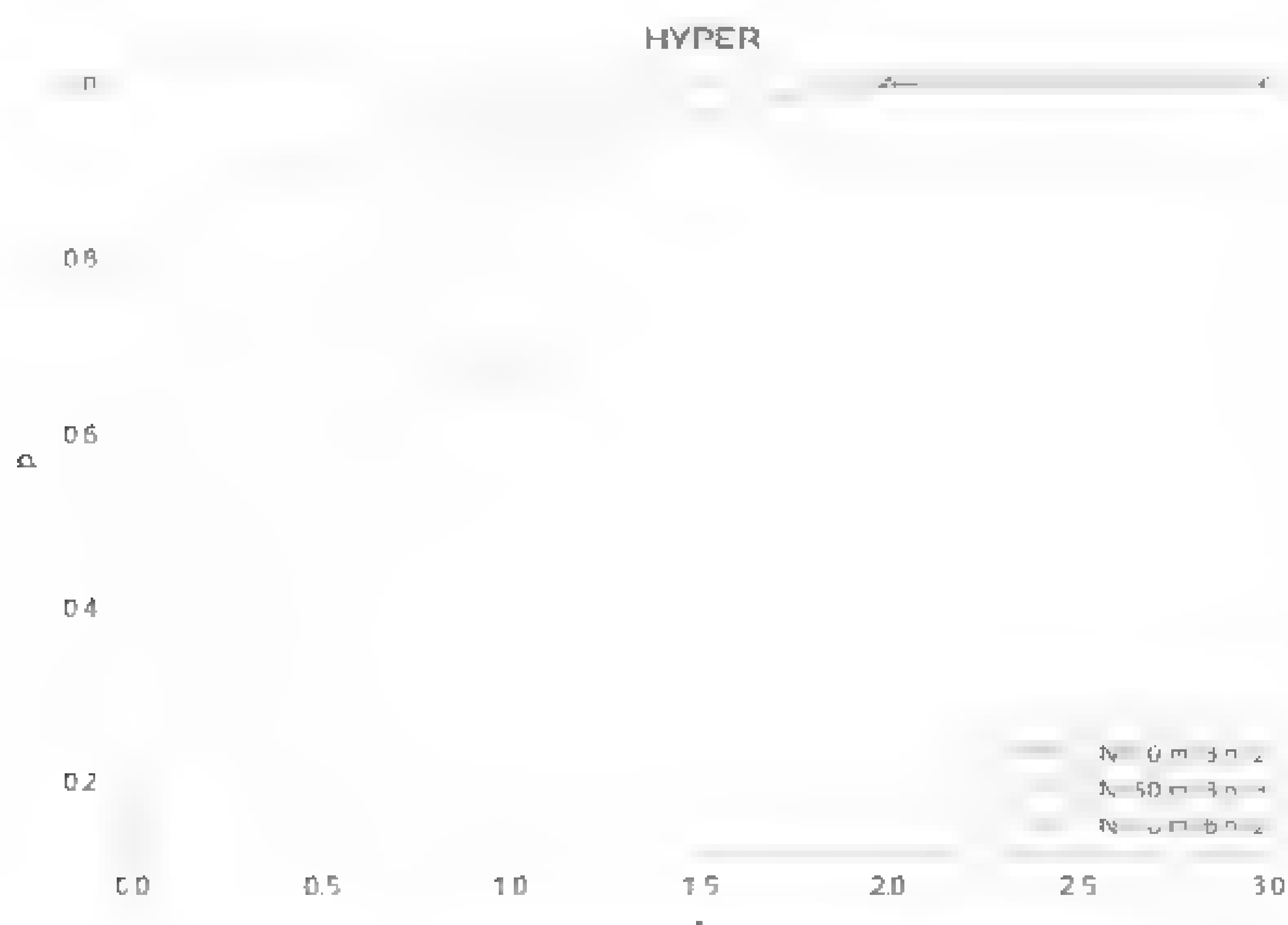


图 20-20 超几何分布的累积分布曲线

20.2.6 泊松分布

二项分布中我们已知，在 N 次试验中获得 n 次成功的概率由如下二项分布的极限形式给出

$$P_p(n|N)=\frac{N!}{n!(N-n)!}P^n(1-p)^{N-n}$$

若将预期成功的试验次数 $\gamma=Np$ 当做自变量，而不再是特定概率 p 下的试验次数 N，则上面的方程变为

$$P_{\gamma/N}(n|N)=\frac{N!}{n!(N-n)!}\left(\frac{\gamma}{N}\right)^n(1-\frac{\gamma}{N})^{N-n}$$

则当试验次数足够大时，其分布就不断逼近如下分布形式：

$$\begin{aligned}P_\gamma(n)&=\lim_{N\to\infty}P_p(n|N)\\&=\lim_{N\to\infty}\frac{N(N-1)\cdots(N-n+1)}{n!}\frac{\gamma^n}{N^n}(1-\frac{\gamma}{N})^N(1-\frac{\gamma}{N})^{-n}\\&=\lim_{N\to\infty}\frac{N(N-1)\cdots(N-n+1)}{N^n}\frac{\gamma^n}{n!}(1-\frac{\gamma}{N})^N(1-\frac{\gamma}{N})^{-n}\\&=1\cdot\frac{\gamma^n}{n!}\cdot e^{-\gamma}\cdot 1\\&=\frac{\gamma^n e^{-\gamma}}{n!}\end{aligned}$$

上面的表达式中，将 γ 和 n 记为 λ 和 x，则有

$$P_\lambda(x)=\frac{\lambda^x e^{-\lambda}}{x!}$$

这就是著名的泊松分布（Poisson Distribution）的数学表达形式，记为 $Po(\lambda)$。注意该分布中试验次数 N 已经完全从概率密度函数中消失，这样对于所有的预期成功的试验次数 γ 就具有相同的函数形式。前面我们已经知道当二项式分布 $\mathrm{Bin}(n,p)$ 在重复次数 n 很大，p 接近于 0 或者 1 时二项分布逼近泊松分布；而 p 不接近于 0 或 1 时二项分布逼近于正态分布。

泊松分布更常见的表达形式是用发生率 λ 表示的，即单位时间内事件发生的次数 γ 比上时间间隔 x，此时 $\lambda=\frac{\gamma}{x}$，即 $\gamma=\lambda x$，则泊松分布的概率密度函数就演变为如下形式：

$$P_\gamma(n)=\frac{(\lambda x)^n e^{-\lambda x}}{n!}$$

泊松分布的均值、方差、偏度和峰度为

$$\begin{aligned}\mu&=\gamma\\\sigma^2&=\gamma\\\gamma_1&=\frac{\mu^3}{\sigma^3}=\frac{\gamma}{\gamma^{3/2}}=\gamma^{-1/2}\\\gamma_2&=\frac{\mu^4}{\sigma^4}-3=\frac{\gamma(1+3\gamma)}{\gamma^2}-3=\frac{\gamma}{\gamma^2}+3-3=\gamma^{-1}\end{aligned}$$

对于上面利用 λ 表示的形式中，x 通常表示为时间区间 t，则 $\lambda=\frac{\gamma}{x}$ 就是 t 时段内发生的次数，记为 $N(t)$，表示 t 时段内事件发生的次数，则泊松分布可表示为

$$P(N(t)=n)=\frac{(\lambda t)^n e^{-\lambda t}}{n!}$$

SAS 程序 20-18 和程序 20-19 绘制泊松分布的概率密度函数曲线和累积分布函数曲线（见图 20-21 和图 20-22）。

程序20-18 泊松分布的概率密度曲线

```
data sample;
  do x=0 to 20;
    p=pdf('POISSON', x, 1);
    p2=pdf('POISSON', x, 4);
    p3=pdf('POISSON', x, 10);
    output;
  end;
run;
proc sgplot;
  title "POISSON";
  series x=x y=p / MARKERS legendlabel="λ=1"
    lineattrs=(pattern=solid color=red);
  series x=x y=p2 / MARKERS legendlabel="λ=4"
    lineattrs=(pattern=dash color=green);
  series x=x y=p3 / MARKERS legendlabel="λ=10"
    lineattrs=(pattern=dot color=blue); ;
  keylegend / location=inside position=topright across=1;
run;
```

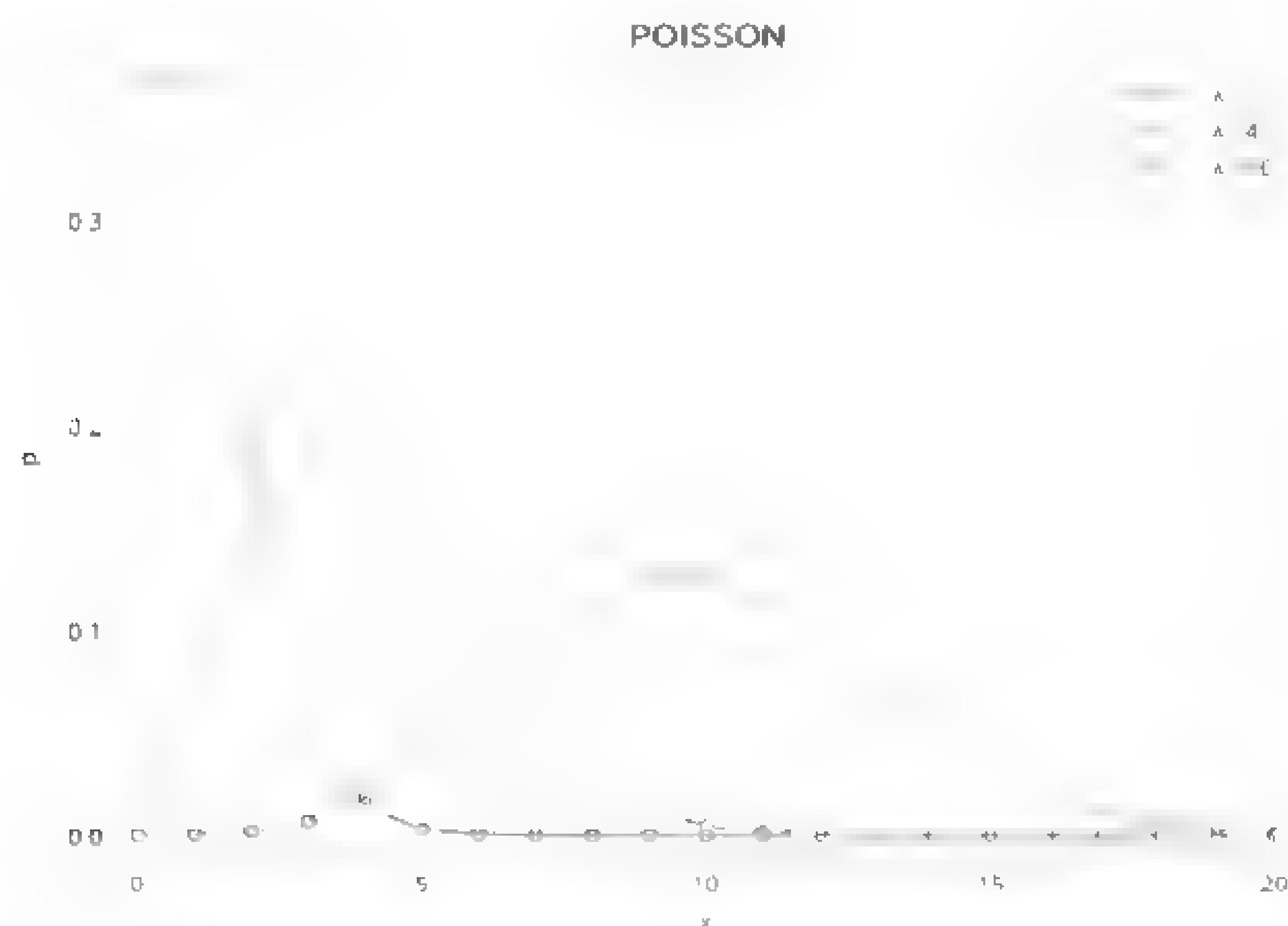


图 20-21 泊松分布的概率密度曲线

程序20-19 泊松分布的累积分布曲线

```
data sample;
  do x=0 to 20  ;
    p=cdf('POISSON', x, 1);
    p2=cdf('POISSON', x, 4);
    p3=cdf('POISSON', x, 10);
    output;
  end;
run;
proc sgplot;
  title "POISSON";
  series x=x y=p / MARKERS legendlabel="λ=1"
    lineattrs=(pattern=solid color=red);
  series x=x y=p2 / MARKERS legendlabel="λ=4"
    lineattrs=(pattern=dash color=green);
  series x=x y=p3 / MARKERS legendlabel="λ=10"
```

```
    lineattrs=(pattern=dot color=blue); ;
  keylegend / location=inside position=bottomright across=1;
run;
```

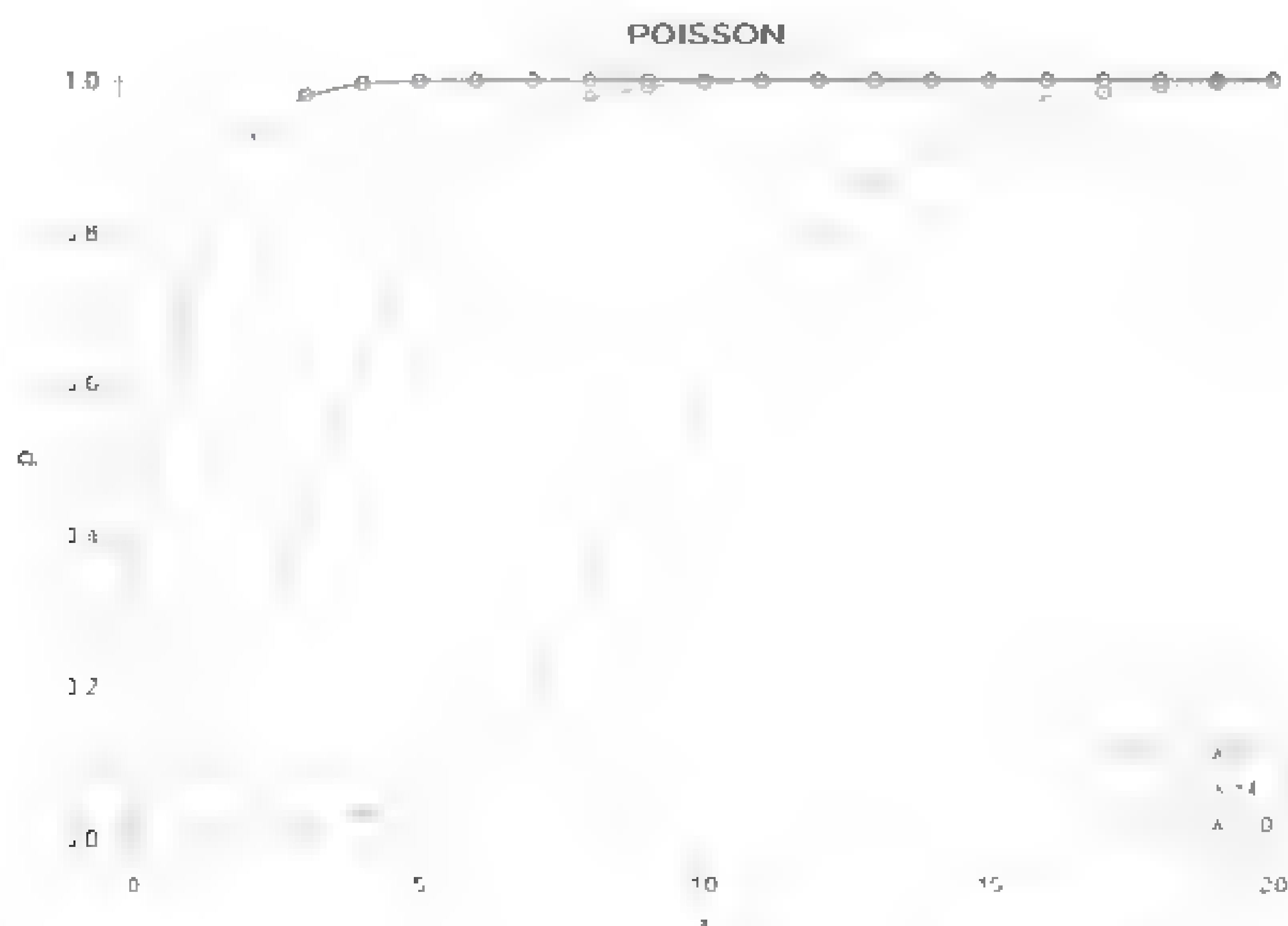


图 20-22　泊松分布的累积分布曲线

从上面的描述可知，泊松分布描述的是单位时间内、独立事件发生次数的概率分布（而指数分布则是独立事件发生时间间隔的概率分布，详见连续型统计分布），它的前提是事件之间不能有关联。

在日常生活中，有大量的事件发生在单位时间内是有固定次数规律的。比如某家医院平均每 60min 就接生 3 个婴儿，某呼叫中心平均 10min 就呼入 1 个电话等。这种有固定发生频率的事件就服从泊松分布。固然我们可以估计在某个时段内发生的次数，但我们永远无法知道发生的具体时间。虽然我们无法精确预测接下来的 1h 内到底有几个婴儿出生，可能是 0 个，也可能是 5 个，但有了泊松分布我们至少可以解决部分确定性问题。

比如已知某家医院平均每 60min 接生 3 个婴儿，在接下来的 2h 内，一个婴儿都不出生的概率是多少？已知信息为每 60min 发生 3 次事件，若以 60min 为一个单位，则事件发生率 $\lambda=3$；则正常状态下 2h（120min）的时段 x 相当于 $x=120/60=2$。代入下面的公式可计算出其概率为 0.2478%。由于它小于小概率事件阈值 1−99.7%=0.3%，也就是说接下来的 2h 内一个婴儿都不出生的概率几乎为 0。即

$$P(N(2)=0)=\frac{(3\times 2)^0 e^{-3\times 2}}{0!}\approx 0.0025$$

在 SAS 的泊松分布的密度函数中，第一个参数 n 为事件预计发生的目标次数，第二个参数 m 则为给定时段 x 内正常发生的次数，它等于发生率 λ 乘以时段倍数 x，即 $\lambda x=3\times(120/60)=6$。上面的例子可用如下 SAS 程序 20-20 进行计算：

程序20-20　特定泊松事件在指定时间段内不发生的概率

```
data _null_;
  lamda=3; /*每60分钟内发生3次试验成功*/

  unit=60;
  x=120/unit;/*目标时段内发生的次数*/
```

```
  m=lamda * x; /*考察 120 分钟内预期发生的次数 m = lamda*x  */

  n=0;  /*120 分钟内一个婴儿都不出生,即 n 0*/
  p=pdf('POISSON', n, m); /*PDF ('POISSON', n, m) */
  put p=;
  if (p<0.003) then put "NOTE: 此为小概率事件(<0.3%) 不可能发生!";
run;
```

泊松分布还可以求解特定时间内事件发生的概率，比如接下来的1小时内至少出生2个婴儿出生的概率。它等于概率总和1.0减去接下来的1小时内出生1个婴儿的概率，再减去接下来的1小时内出生0个婴儿的概率，结果约为80%，即

$$P(N(1)\geqslant 2)=1-P(N(1)=1)-P(N(1)=0)$$
$$=1-\frac{(3\times1)^1 e^{-3\times1}}{1!}-\frac{(3\times1)^0 e^{-3\times1}}{0!}\approx 0.8009$$

本书附录2列出了常用的泊松分布累积概率表，可用于直接查表得到累积概率值。也可用于如下SAS 程序直接求解（见程序20-21）。

程序20-21　特定泊松事件在指定时间段内发生的概率

```
data _null_;
  lamda=3; /*每60分钟内发生3次试验成功*/
  t0=60;

  n=0;tx=60;
  x=tx/t0;
  m=lamda * x;
  p0=pdf('POISSON', n, m); /*1小时内生产0个小孩的概率*/

  n=1;tx=60;
  x=tx/t0;
  m=lamda * x;
  p1=pdf('POISSON', n, m); /*1小时内生产1个小孩的概率*/

  p=1-p1-p0; /*1小时内至少生产2个以上小孩的概率*/
  put p= ;

  if (p>0.9973)      then put "NOTE: 此为大概率事件(>99.7%) 极有可能发生!";
  else if (p>0.954) then put "NOTE: 此为大概率事件(>95%)   很有可能发生!";
  else if (p>0.683) then put "NOTE: 此为大概率事件(>68%)   有可能发生!";
  else if (p<0.003) then put "NOTE: 此为小概率事件(<0.3%)  即不可能发生!";
run;
```

20.3　连续型统计分布

如果随机变量取值是连续的，则对应统计分布为连续型统计分布，连续型分布主要包括以下几种。

20.3.1　正态分布

正态分布 (Normal Distribution) 是最著名的连续型统计分布，也叫高斯分布。它的定

义是：若连续型变量X服从如下概率密度函数所定义的统计分布，则称变量X服从均值为μ、方差为σ^2的正态分布，通常记为$N(\mu,\sigma^2)$，其概率密度函数为

$$P(x)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

其中x介于负无穷到正无穷之间，与各种概率密度函数一样，正态分布概率密度函数在负无穷到正无穷上的积分为1.0，即

$$\int_{-\infty}^{\infty}P(x)\mathrm{d}x=1$$

而对所有小于等于的概率进行累积求积分，则得到累积分布函数：

$$D(x)=\int_{-\infty}^{x}P(x')\mathrm{d}x'=\frac{1}{\sigma\sqrt{2\pi}}\int_{-\infty}^{x}e^{-(x'-\mu)^2/(2\sigma^2)}\mathrm{d}x'$$

正态分布的累积分布函数也可以写为

$$D(x)=\frac{1}{2}\left[1+\mathrm{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right]$$

其中误差函数erf定义为$\mathrm{erf}(x)=\frac{2}{\sqrt{\pi}}\int_{x}^{\infty}e^{-t^2}\mathrm{d}t$

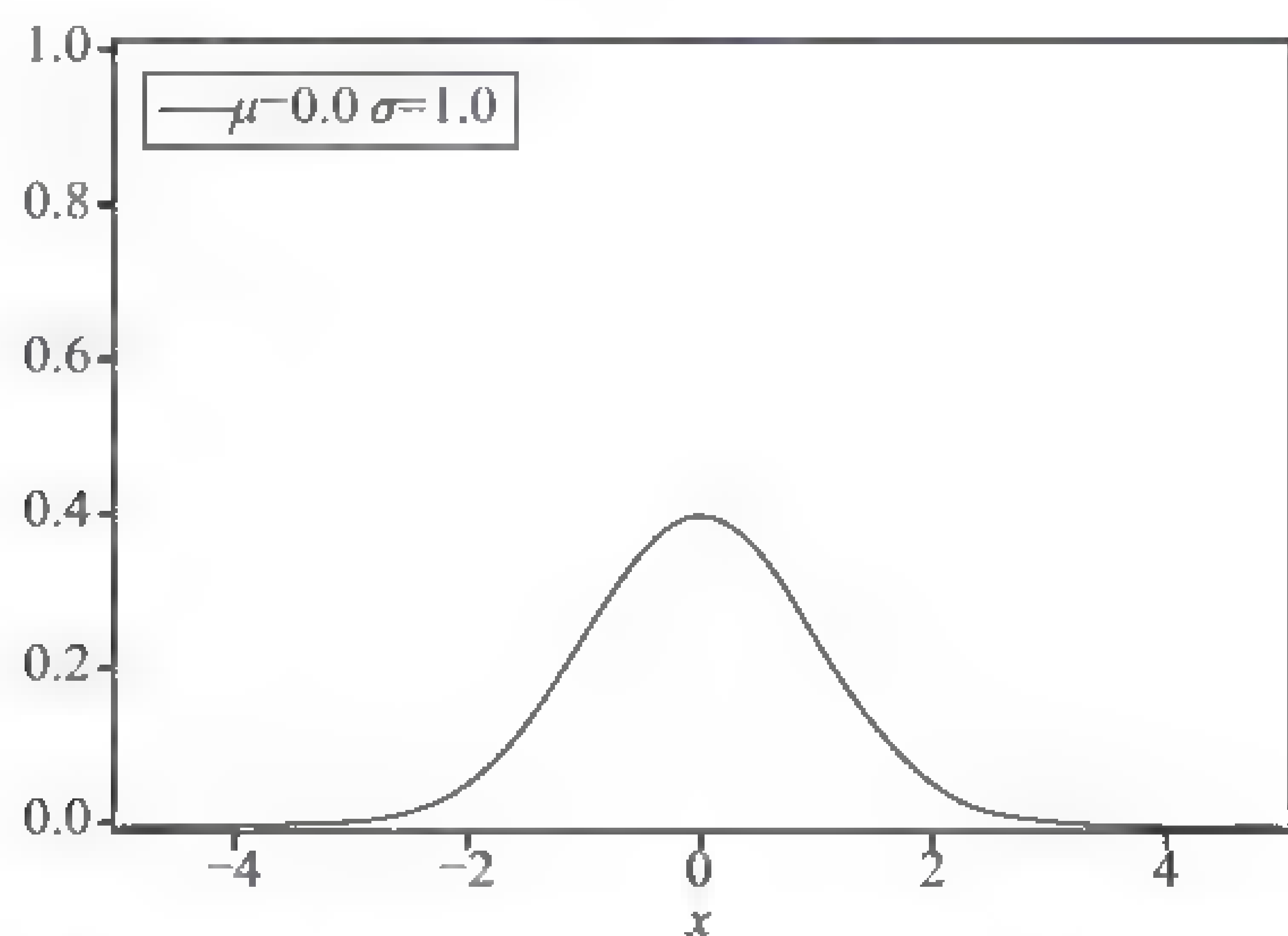


图20-23 标准正态分布的概率密度曲线（钟形曲线）

正态分布的均值为μ，它决定了统计分布的位置，而其标准差σ则决定了分布的区间范围。正态分布因为中心极限定理的存在而非常有用，因为对于独立分布的随机变量，观测样本的平均值在观测数量足够大时，其分布将近似正态分布。正态分布是许多连续和离散统计分布的极限分布，也就是说，因为中心极限定理的结论，其他概率分布可以用正态分布作近似。比如在n相当大，p接近于0.5时二项分布$\mathrm{Bin}(n,p)$近似正态分布$N(np,np(1-p))$，而泊松分布$Po(\lambda)$在当取样样本数很大时近似正态分布$N(\lambda,\lambda)$。

正态分布最早是由法国数学家棣莫弗在逼近二项分布的过程中发现的，随后1783年被拉普拉斯用于研究计量误差，并在1809年由高斯用于分析天文学数据。正态分布是该统计分布在统计学上的名称，在物理学领域它被称为高斯分布，在社会科学领域则被称为钟形曲线。然而诸多统计分布，如柯西分布、学生t分布和对数分布也都呈现钟

形曲线形态，因此钟形曲线并非正态分布所独有的分布形态，需要特别注意这一点。

1. 标准正态分布

总体均值$\mu=0$、方差为$\sigma^2=1$的正态分布称为标准正态分布。此时概率密度函数和累积分布函数简化为

$$P(x)=\frac{e^{-\frac{1}{2}x^2}}{\sqrt{2\pi}}$$

$$D(x)=\frac{1}{2}\left[1+\operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right]$$

实际上，任何一般正态分布都可以通过平移缩放变换$Z=\frac{X-\mu}{\sigma}$，使得变换后的随机变量Z服从标准正态分布$N(0, 1)$，标准正态分布概率密度函数中的分母$\sqrt{2\pi}$确保了钟形曲线下方的面积为1，而指数系数1/2则是为了确保方差或标准差为1。标准正态分布曲线是关于$x=0$左右对称的函数，其最大值为$\frac{1}{\sqrt{2\pi}}=0.3989$，且$x$等于正负1时为1个$\sigma$拐点。下面的SAS程序是绘制标准正态分布曲线的方法之一（见程序20-22及其输出图20-24）。

程序20-22　绘制标准正态分布曲线

```
data mydata;
  do x=-4 to 4 by 0.01;
    y=pdf("NORMAL", x);
    output;
  end;
run;
proc sgplot data=mydata;
  title " ";
  series x=x y=y;
  yaxis max=1 ;
  refline 0 /axis=x lineattrs=(pattern=2)    ;
  refline 0.3989 / lineattrs=(pattern=2) label="0.3989";
run;
```



图20-24　标准正态分布

同理，如果Z是一个标准正态分布，则$X=Z\sigma+\mu$必定服从期望为μ，标准差为σ的一般正态分布$N(\mu, \sigma^2)$。一般地，正态分布图波峰位置所对应的横坐标是总体均值μ，

标准差 σ 越大，钟形曲线越矮，越往两侧延伸，表示分散在期望 μ 两侧的数据越多，数据越离散。

服从正态分布的数据有 3σ 准则，即 68 95 99.7 法则（见图 20-25）：该准则指出如果变量分布服从一般正态分布 $N(\mu, \sigma^2)$，则变量观测值有 68% 的概率落在总体均值 μ 两侧 1σ 的区间内，95% 的概率落在总体均值 μ 两侧 2σ 的区间内；99.7% 的概率落在总体均值 μ 两侧 3σ 区间内。落在 3σ 之外的概率约为 1 99.7% 0.3%，通常称之为小概率事件，而小概率事件通常不发生。

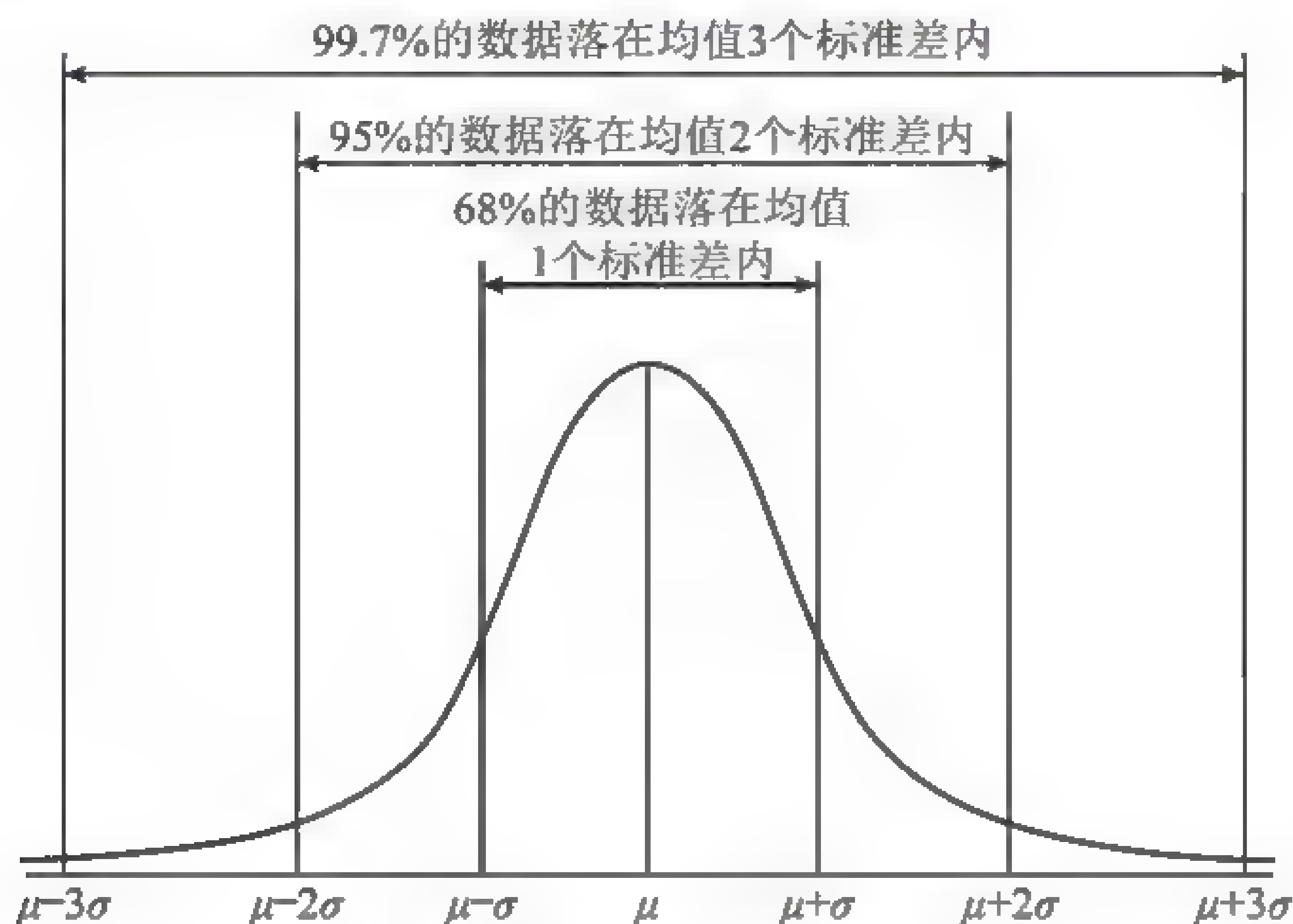


图 20-25　正态分布的 3σ 准则

正态分布的 3σ 准则是如何算出来的呢？由于正态分布是连续型分布，因此要计算正态分布曲线下方小于 $\mu-3\sigma$ 和大于 $\mu+3\sigma$ 的面积，根据对称性，我们只需要计算一侧即可。为简化问题，我们使用标准正态分布 $N(0, 1)$ 为例子，其概率密度函数简化为

$$P(x)=\frac{e^{-\frac{1}{2}x^2}}{\sqrt{2\pi}}$$

要计算 $x=(\mu-3\sigma)=-3$ 左侧的面积，可用积分思想（见图 20-26）来计算，即用一系列的小矩形面积来逼近函数曲线下方的面积，其计算代码见程序 20-23 所示。

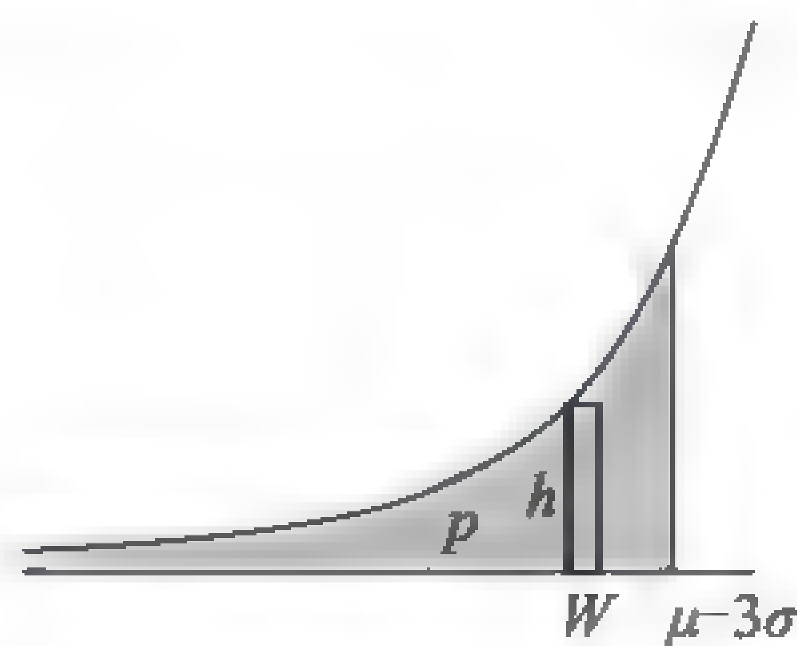


图 20-26　积分计算 3σ 准则值

程序20-23 积分求解3σ准则值

```
data _null_;
  x0=-3; /*3 西格玛*/
  p=0; w=0.01;
  do x=-37.6 to x0 by w;
    h=(constant("E")** (-x*x/2.0)) / sqrt( 2 * constant("PI"));
    p=p+ w * h;
  end;
  q=1-2*p;
  put p= 6.4 q= 6.4 ;
run;
```

系统输出 p=0.0014 q=0.9973，即 3σ 个内的概率为 99.73%。在 SAS 中也可以直接调用通用的概率密度函数 pdf 进行计算，即将上面的 h=(constant("E")** (-x*x/2.0))/sqrt(2 * constant("PI")) 一行改为 h=pdf("NORMAL", x) 即可。在 SAS 中还有一种方法是直接调用 SAS 系统的 CDF 累积分布函数进行计算（见程序 20-24），结果相同。

程序20-24 正态分布CDF 求解3σ准则值

```
data _null_;
  p=cdf("NORMAL", -3); /*3西格玛*/
  q=1-2*p;
  put q= 6.4;
run;
```

2. 正态分布的性质

数学上现在已经证明，正态分布是二项分布参数 n（N 重伯努利分布成功的次数）变大时的极限情况。当二项分布的 n 足够大时，二项分布逼近均值为 np，方差为 $np(1-p)$ 的正态分布 $N(np, np(1-p))$，即 $\mu=np, \sigma^2=npq=np(1-p)$。

正态分布有很多奇特的属性，实践中未知分布的随机变量通常可假设服从正态分布，尤其是物理学和天文学的计算。尽管该假设可能有危险，但由于中心极限定理的存在，我们通常能得到很好的逼近结果，并且往往证明该假设是一个很好的近似。中心极限定理指出具有有限均值和方差的任何分布之任何变量观测集，变量的均值趋于正态分布。日常生活中如考试成绩、身高等都大致遵从正态分布，少数成员在高低两端，大多数成员在中间区间聚集。

正态分布是无限可分且稳定的最大熵概率分布。通常系统中如果被观察的对象之间相互独立，相关性很弱，则对象在系统中的分布根据中心极限定理往往服从正态分布。虽然有各种各样的统计分布，但正态分布似乎才是统计分布中的王者，作者在附录中给出的图表完整展示了各种统计分布之间的相互关系以及与正态分布的联系。

假如随机变量 X 和 Y 变量来自两个具有不同均值和方差的独立正态分布，它们的和与差依然服从正态分布。服从正态分布的 X 与 Y 的比则服从柯西分布（Cauchy Distribution），服从标准正态分布的 X 与 Y 的比率，则服从标准柯西分布；柯西分布是正态分布的姊妹分布。

正态分布的概率密度函数和累积分布函数可用 SAS 代码（见程序 20-25 和程序 20-26）绘制如下，其结果如图 20-27 和图 20-28 所示。

程序20-25 正态分布的概率密度曲线

```
data sample;
  do x =-5 to 5 by 0.01 ;
    p=pdf('NORMAL', x,   0,0.2);
    p2=pdf('NORMAL', x,  0,1.0);
    p3=pdf('NORMAL', x,  0,5.0);
    p4=pdf('NORMAL', x, -2,0.5);
    output;
  end;
run;
proc sgplot;
  title "NORMAL";
  series x=x y=p /  legendlabel="μ= 0.0  σ=0.2"
    lineattrs=(pattern=solid color=red);
  series x=x y=p2 /  legendlabel="μ= 0.0  σ=1.0"
    lineattrs=(pattern=dash color=green);
  series x=x y=p3 /  legendlabel="μ= 0.0  σ=5.0"
    lineattrs=(pattern=dot color=blue); ;
  series x=x y=p4 /  legendlabel="μ=-2.0  σ=0.5"
    lineattrs=(pattern=dashdashdot color=black); ;
  keylegend / location=inside position=topright across=1;
run;
```

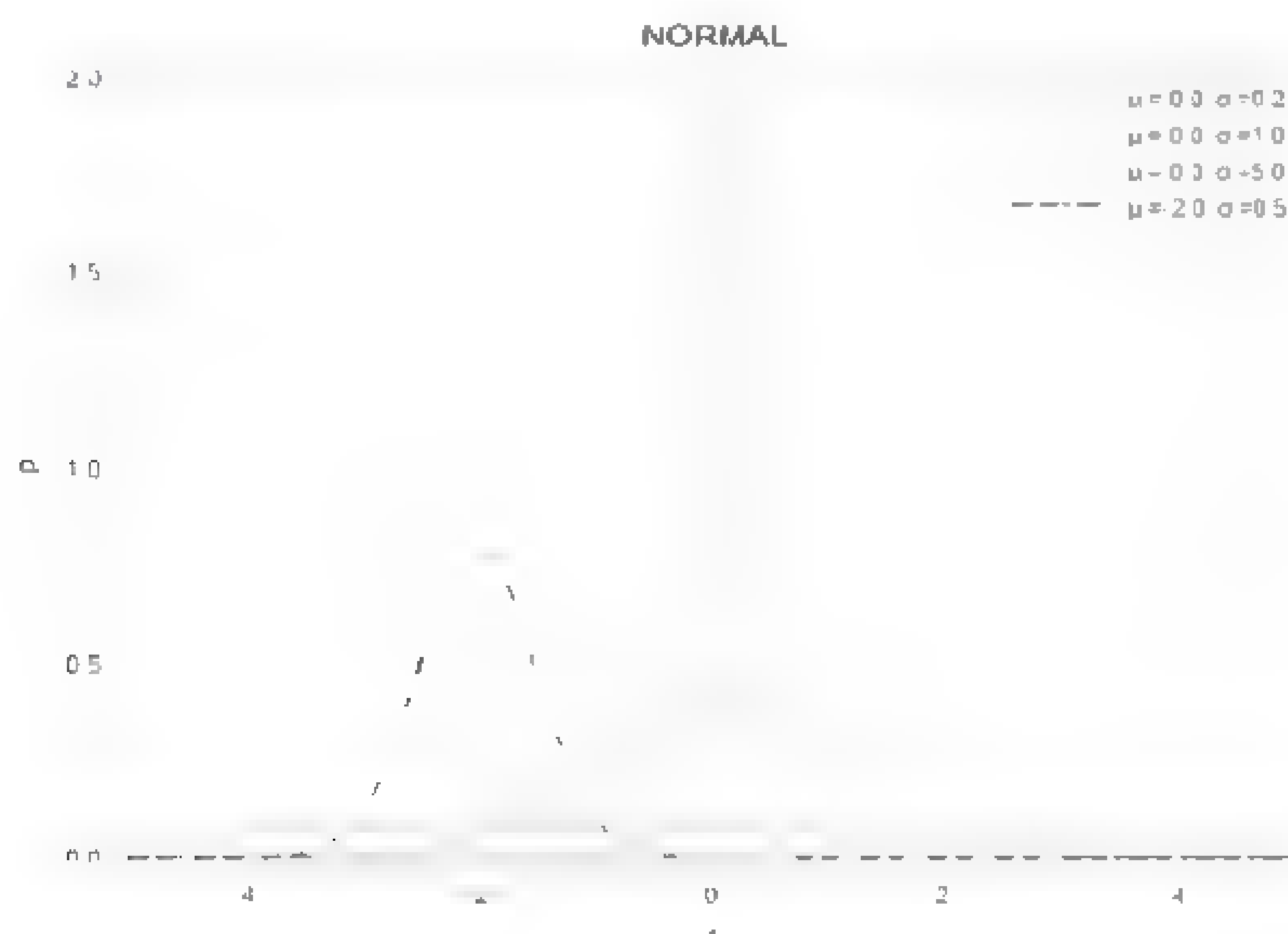


图 20-27 正态分布的概率密度曲线

程序20-26 正态分布的累积分布曲线

```
data sample;
  do x=-5 to 5 by 0.01 ;
    p=cdf('NORMAL', x,   0,0.2);
    p2=cdf('NORMAL', x,  0,1.0);
    p3=cdf('NORMAL', x,  0,5.0);
    p4=cdf('NORMAL', x, -2,0.5);
    output;
  end;
run;
proc sgplot;
  title "NORMAL";
  series x=x y=p / legendlabel="μ= 0.0  σ=0.2"
    lineattrs=(pattern=solid color=red);
  series x=x y=p2 / legendlabel="μ= 0.0  σ=1.0"
    lineattrs=(pattern=dash color=green);
  series x=x y=p3 / legendlabel="μ= 0.0  σ=5.0"
```

```
    lineattrs=(pattern=dot color=blue); ;
  series x=x y=p4 / legendlabel="μ=-2.0 σ=0.5"
    lineattrs=(pattern=dashdashdot color=black); ;
  keylegend / location=inside position=bottomright across=1;
run;
```

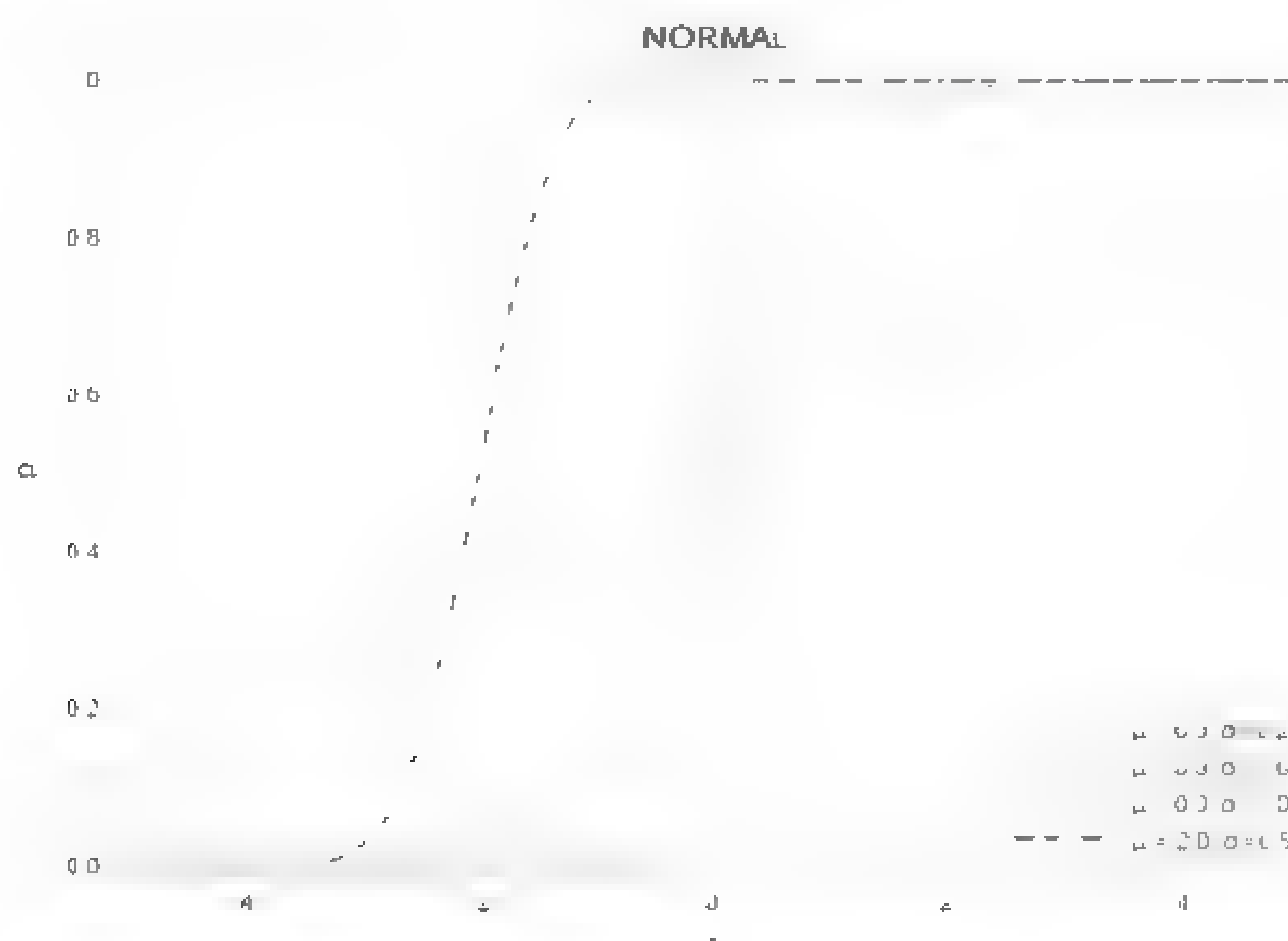


图 20-28 正态分布的累积分布曲线

对于任何一个给定样本数据，检查某个变量是否服从正态分布可用 Q-Q 图进行检查。为了说明问题，我们首先用随机数函数生成一个包含 10000 个服从正态分布的数据样本 x（见程序 20-27），随后用 SAS 检查其正态性并进行验证。

程序20-27 生成服从正态分布的随机数样本

```
data sample;
  do i = 1 to 10000;
    x= rand('NORMAL') ;
    output;
  end;
  keep x ;
run;
```

然后我们调用 PROC SGPLOT 绘制直方图和密度曲线检查变量 x 的分布特征，如果核密度曲线和正态分布曲线很贴近，说明数据服从正态分布（见程序 20-28 和图 20-29）。

程序20-28 直方图和核密度估计检查随机数样本的分布特征

```
title;
proc sgplot data=sample;
  histogram x;
  density x /  type=kernel;
  density x /  type=normal ;
run;
```

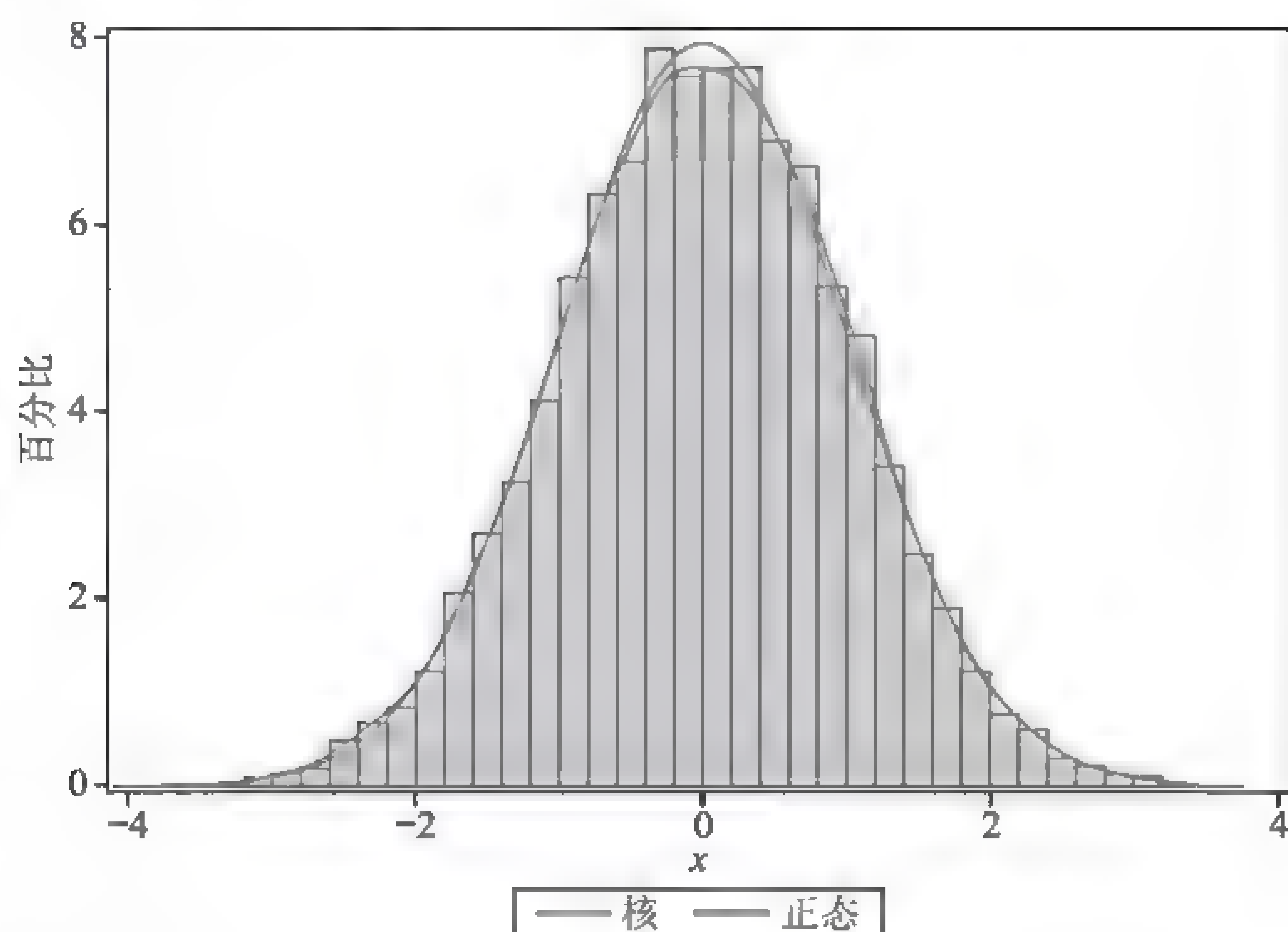


图 20-29　核密度曲线与正态分布曲线

更多的情况是用 Q-Q 图进行检验，如果数据在 Q-Q 图上的投点形成一条 45° 角的直线，说明该样本数据服从正态分布的。在 SAS 中可以用多种方法绘制 Q-Q 图，下面以 PROC UNIVARIATE 为例（见程序 20-29 和输出图 20-30）。

程序20-29　Q-Q图正态性检验
```
proc univariate data=sample;
  var x;
  qqplot x ;
run;
```

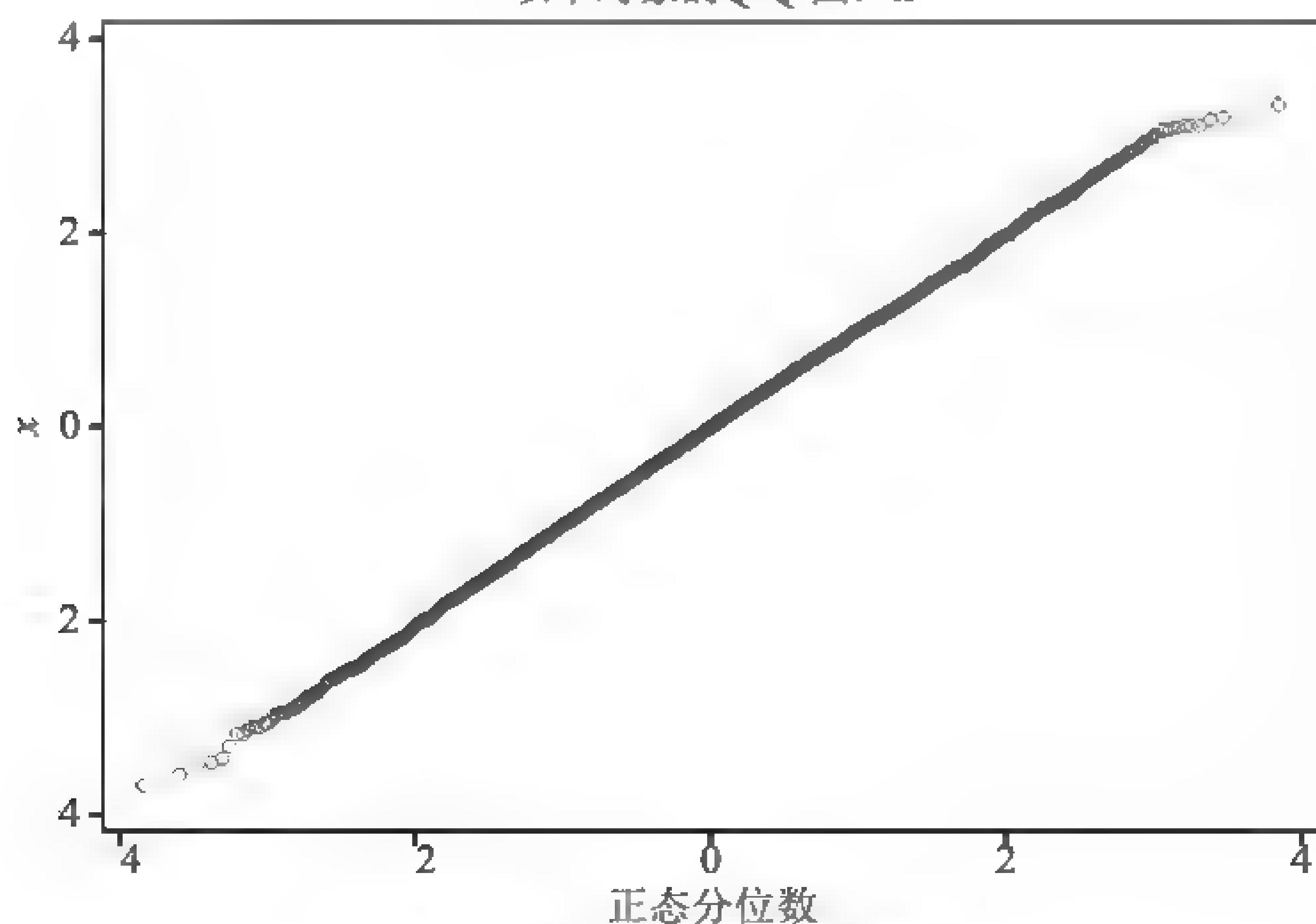


图 20-30　Q-Q 图检验正态性

假如前面的程序 20-27 用 x　rand('UNIFORM') 随机数函数生成样本数据集，此时

数据服从的是均匀分布。再用前面的 PROC SGPLOT 和 Q-Q 图检验其分布特征，将会得到如图 20-31 所示的分布：其核密度曲线是一个矩形。而 Q-Q 图则表现为一个 S 函数的形状，这些特征与正态分布的图形具有显著区别。

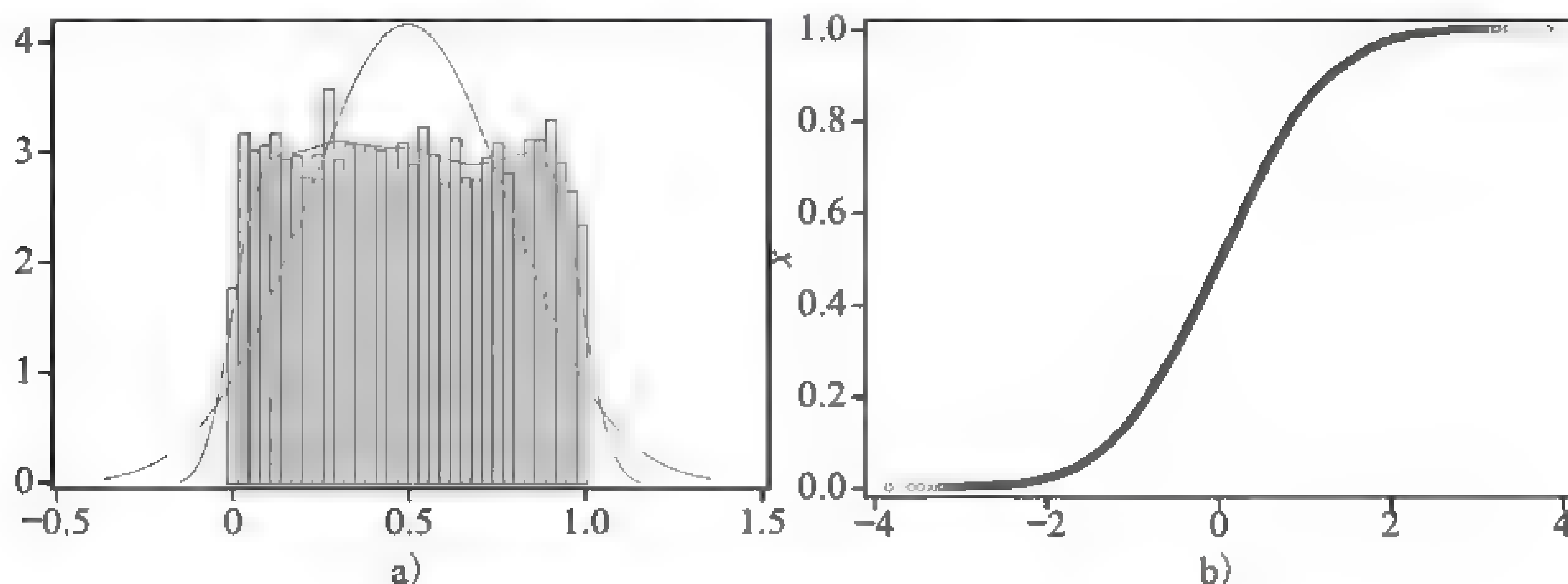


图 20-31　均匀分布的核密度曲线和 Q-Q 图

20.3.2　对数正态分布

如果连续变量的对数 ln(X) 服从正态分布，则连续变量 X 服从对数正态分布 (Log-Normal)，通常记为 $X \sim LnN(\mu,\sigma^2)$ 。其概率密度函数和累积分布函数与正态分布相同，但变量为 ln(x) 而非 x。对数正态分布的均值、方差、偏度系数和峰度系数为

$$\mu = e^{M+\frac{S^2}{2}}$$
$$\sigma^2 = e^{S^2+2M}(e^{S^2}-1)$$
$$\gamma_1 = \sqrt{e^{S^2}-1}\,(2+e^{S^2})$$
$$\gamma_2 = e^{4S^2}+2e^{3S^2}+3e^{2S^2}-6$$

其中 M 和 S^2 为变量对数的均值和方差。生活中服从对数正态分布变量的例子包括消毒剂中细菌的存活时间，照相技术感光乳剂中的银粒子大小、人的体重和血压，以及书籍中一个句子中的单词个数等。对数正态分布和正态分布之间存在一个可逆的变量转换关系 ln(x) 和 e^x。程序 20-30 验证了对数正态分布和正态分布之间的关系，其输出如图 20-32 所示。

程序20-30　对数正态分布与正态分布之间的转换

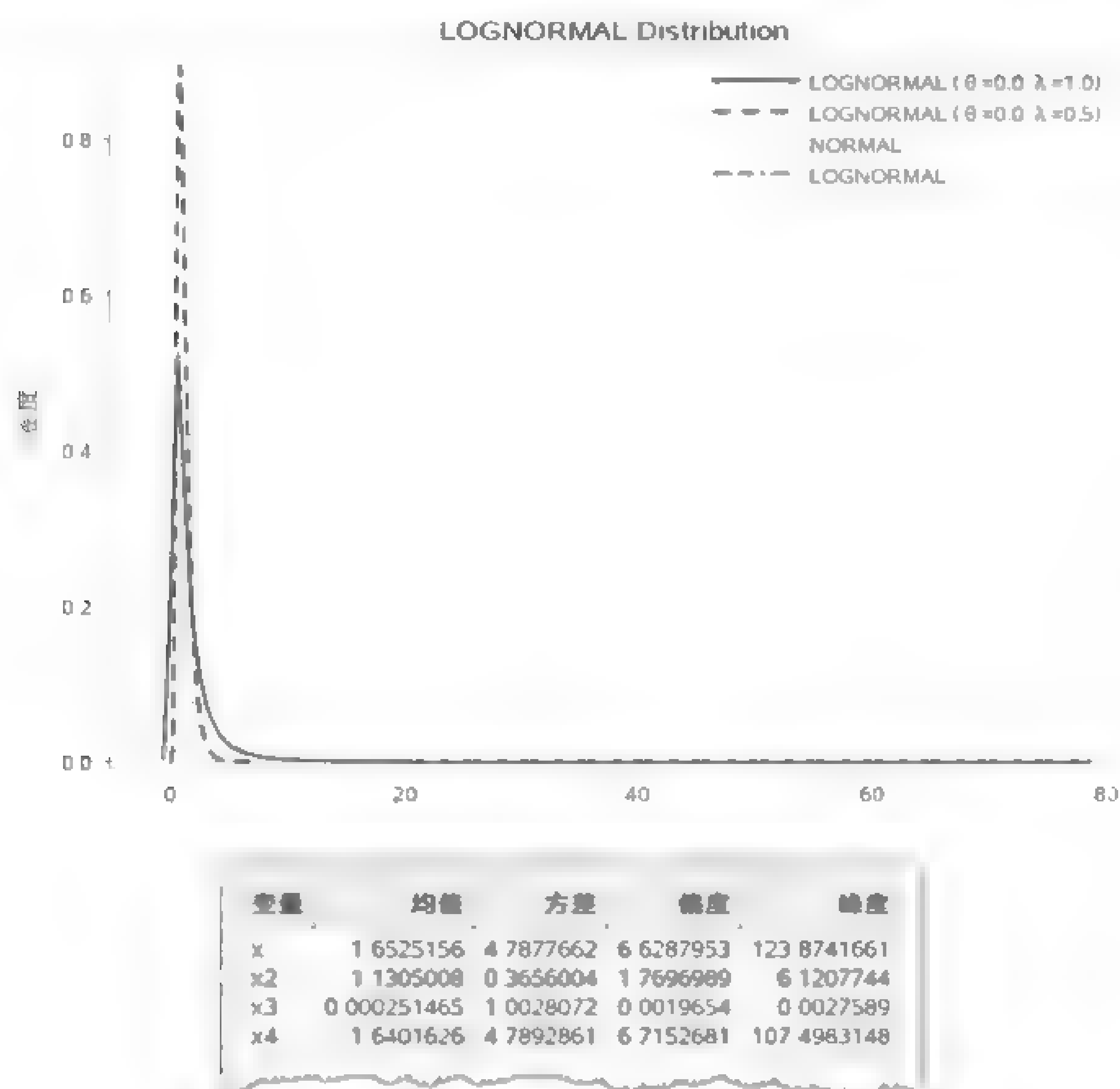
```
data sample;
  do i = 1 to 100000;
    x=RAND('LOGNORMAL');

    theta=0; lamda=0.5;
    x2=RAND('LOGNORMAL',theta, lamda);

    x3=log(x);

    x4=exp(rand('NORMAL'));
    output;
  end;
```

```
  keep x x2 x3 x4 ;
run;
proc sgplot data=sample;
  title 'LOGNORMAL Distribution ';
  density x / type=kernel legendlabel='LOGNORMAL (θ=0.0 λ=1.0)'
    lineattrs=(pattern=solid color=blue);
  density x2 / type=kernel legendlabel='LOGNORMAL (θ=0.0 λ=0.5)'
    lineattrs=(pattern=dash color=blue);
  density x3   / type=kernel legendlabel='NORMAL'
    lineattrs=(pattern=dot);
  density x4   / type=kernel legendlabel='LOGNORMAL'
    lineattrs=(pattern=dashdashdot);
  keylegend / location=inside position=topright across=1;
  xaxis display=(nolabel);
run;
proc means data=sample mean var skewness kurtosis; run;
```



变量	均值	方差	偏度	峰度
x	1.6525156	4.7877662	6.6287953	123.8741661
x2	1.1305008	0.3656004	1.7696989	6.1207744
x3	0.000251465	1.0028072	0.0019654	0.0027589
x4	1.6401626	4.7892861	6.7152681	107.4983148

图 20-32　对数正态分布概率密度曲线与期望

从上面的表中可以看出，x3 服从标准正态分布 N(0, 1)，而 x4 服从对数正态分布。为了进一步验证这个，我们运行程序 20-31 查看结果，可以看到服从对数正态分布的变量，其对数值服从正态分布（见图 20-33）。

程序20-31　对数正态分布与正态分布

```
proc sgplot data=sample;
  title 'LOGNORMAL Distribution ';
  density x3 / type=normal legendlabel='NORMAL '
    lineattrs=(pattern=solid color=blue);
  density x3   / type=kernel legendlabel='Log(x): x ~ LOGNORMAL'
    lineattrs=(pattern=solid);

  keylegend / location=inside position=topright across=1;
```

```
  xaxis display=(nolabel);
run;
```

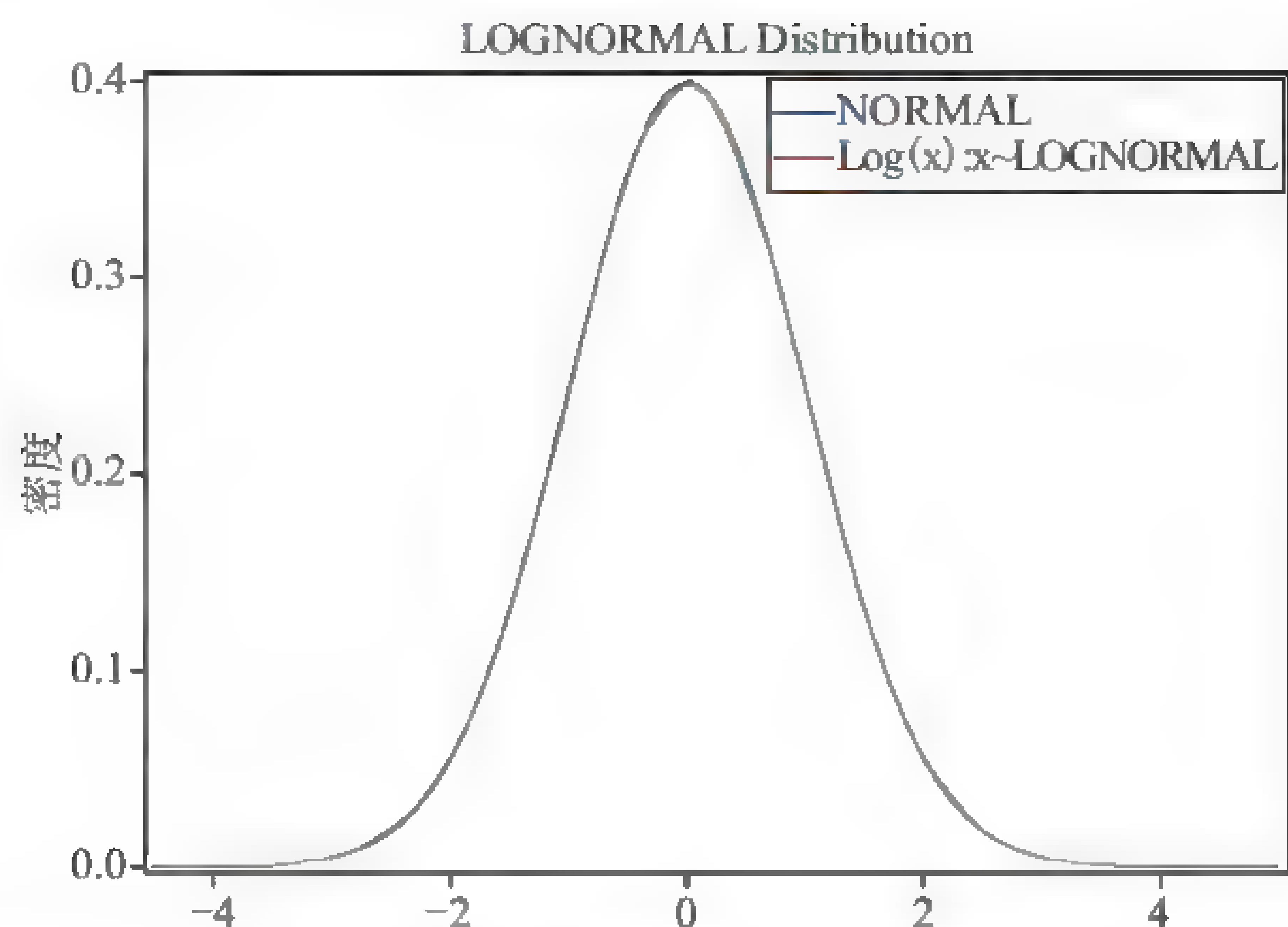


图 20-33　对数正态分布与正态分布

同理，我们也可以用程序 20-32 进行验证：服从正态分布的变量，其 exp(x) 变换服从对数正态分布（见图 20-34）。

程序20-32　正态分布指数变换后服从对数正态分布

```
proc sgplot data=sample;
  title 'LOGNORMAL Distribution ';
  density x   / type=kernel legendlabel='LOGNORMAL '
    lineattrs=(pattern=solid color=blue);
  density x4  / type=kernel legendlabel='exp(x): x ~ NORMAL'
    lineattrs=(pattern=solid);
  keylegend / location=inside position=topright across=1;
  xaxis display=(nolabel);
run;
```

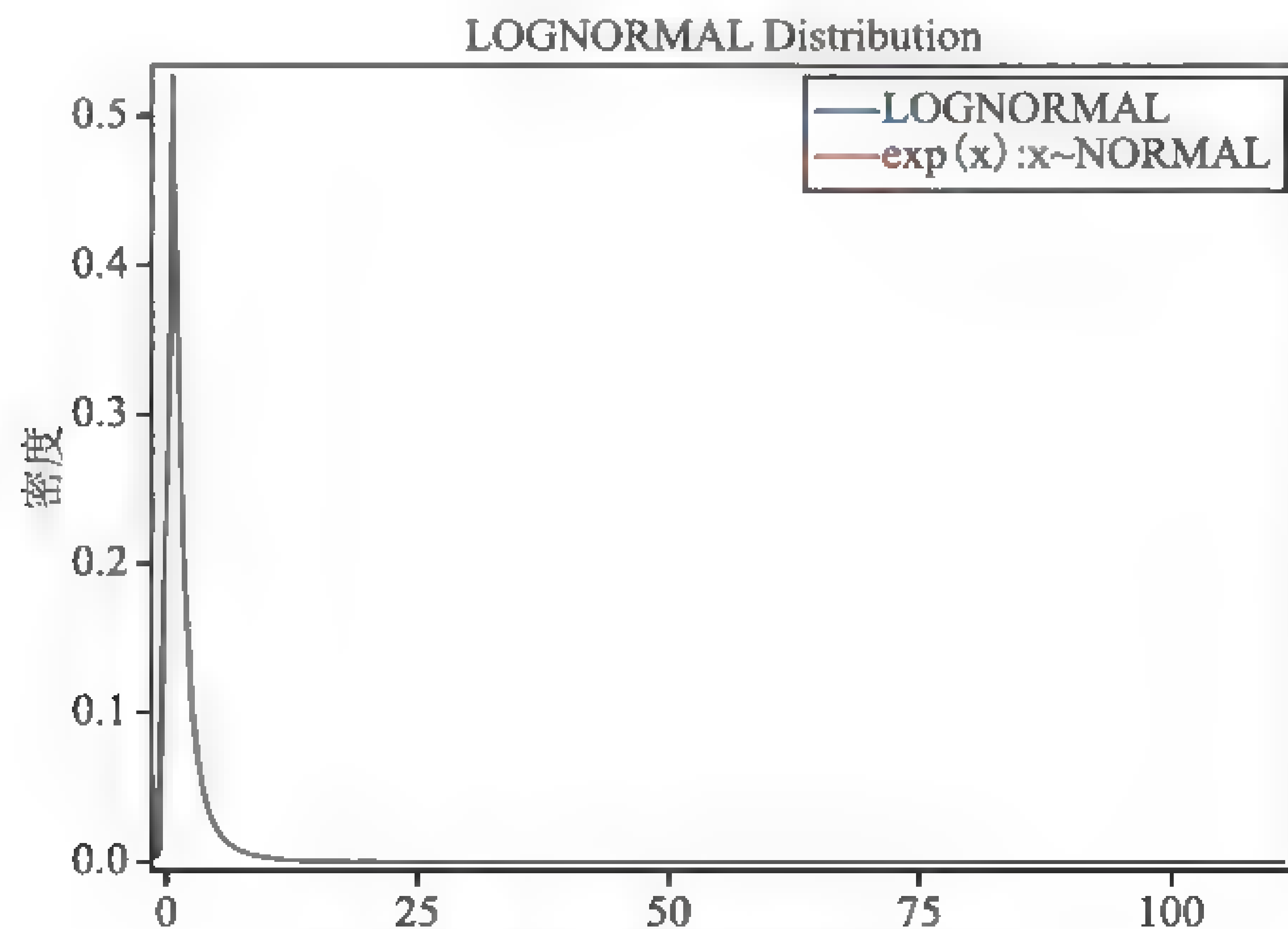


图 20-34　正态分布与对数正态分布

20.3.3 指数分布

对于给定泊松分布，若其事件的发生率为λ，则其后事件连续发生之间的等待时间，遵循如下概率密度函数描述的统计分布，称为指数分布（Exponential Distribution）（见图 20-35）。

$$P(x)=D'(x)=\lambda e^{-\lambda x}$$

$$D(x)=P(X\leqslant x)=1-P(X>x)=1-e^{-\lambda x}$$

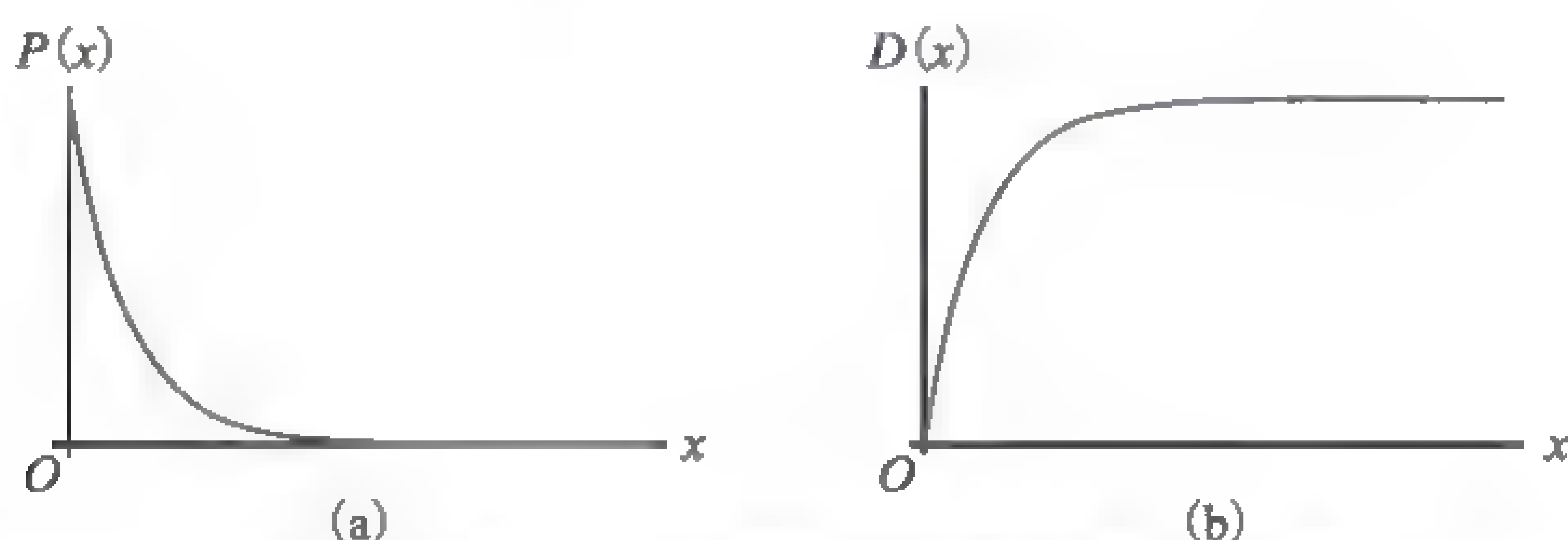


图 20-35 指数分布的概率密度曲线与累积分布曲线

非负实数的指数分布是连续型统计分布中唯一的无记忆（Memoryless）随机分布，而非负整数的几何分布则是离散型统计分布唯一的无记忆分布。在统计学上"无记忆"是指概率分布的一种属性，是指与系统状态无关，事件发生概率不受历史过程的影响。即特定事件发生之前的等待时间不取决于已经过去了多少时间。指数分布的均值、方差、偏度和峰度为

$$\mu=\frac{1}{\lambda}$$

$$\sigma^2=\frac{1}{\lambda^2}$$

$$\gamma_1=2$$

$$\gamma_2=6$$

也可记为$X\sim \mathrm{Exp}(\beta)$，其中$\beta=1/\lambda$，就记为如下形式，此时指数分布的均值$\mu=\beta$，方差$\sigma^2=\beta^2$

$$P(x)=\frac{1}{\beta}e^{-x/\beta}$$

$$D(x)=1-e^{-x/\beta}$$

与离散泊松分布不同，指数分布是泊松事件的时间间隔的概率，从概率密度曲线$P(x)$可以看出，泊松事件发生的概率随着时间间隔的增长呈指数式衰减。日常生活中比如某家医院婴儿出生的时间间隔、呼叫中心呼入电话之间的时间间隔等都服从指数分布。SAS 程序 20-33 和程序 20-34 可绘制指数分布的概率密度曲线和累积分布曲线（见图 20-36 和图 20-37）。

程序20-33 指数分布的概率密度曲线

```
data sample;
  do x=0 to 5 by 0.01;
    p=pdf("Exponential", x,   0.5);
    p2=pdf("Exponential", x,  1);
    p3=pdf("Exponential", x,  2.5);
```

```
    output;
  end;
run;
proc sgplot;
  title "Exponential";
  series x=x y=p  /  legendlabel="β=0.5"
    lineattrs=(pattern=solid color=red);
  series x=x y=p2 /  legendlabel="β=1 "
    lineattrs=(pattern=solid color=green);
  series x=x y=p3 /  legendlabel="β=2.5"
    lineattrs=(pattern=solid color=blue); ;

  keylegend / location=inside position=topright across=1;
  yaxis max=2;
run;
```

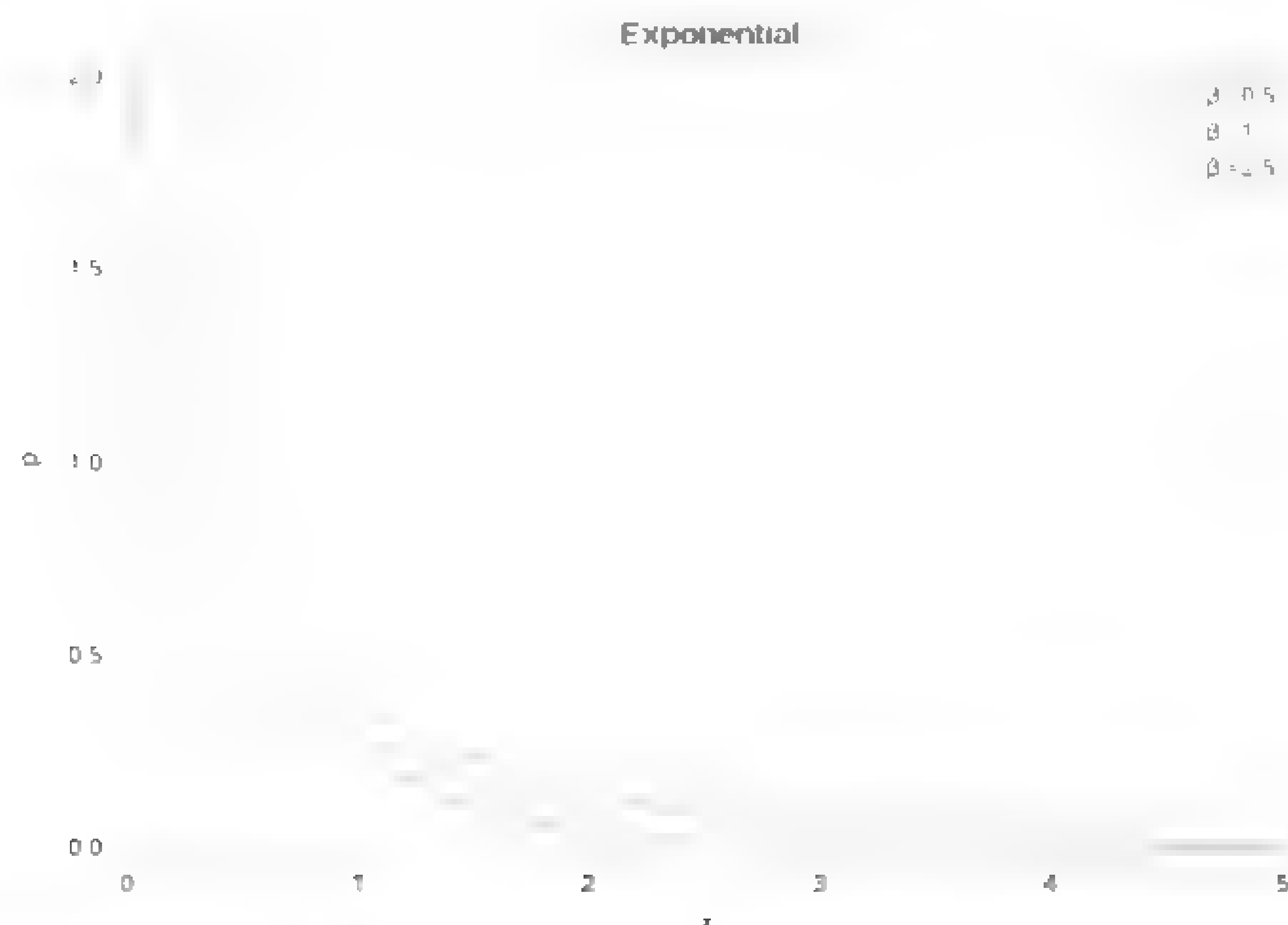


图 20-36　指数分布的概率密度曲线

程序20-34　指数分布的累积分布曲线

```
data sample;
  do x=0 to 5 by 0.01 ;
    p=cdf("Exponential", x,   0.5);
    p2=cdf("Exponential", x,  1);
    p3=cdf("Exponential", x,  2.5);
    output;
  end;
run;
proc sgplot;
  title "Exponential";
  series x=x y=p /  legendlabel="β=0.5"
    lineattrs=(pattern=solid color=red);
  series x=x y=p2 /  legendlabel="β=1 "
    lineattrs=(pattern=dash color=green);
  series x=x y=p3 /  legendlabel="β=2.5"
    lineattrs=(pattern=dot color=blue); ;

  keylegend / location=inside position=bottomright across=1;
run;
```

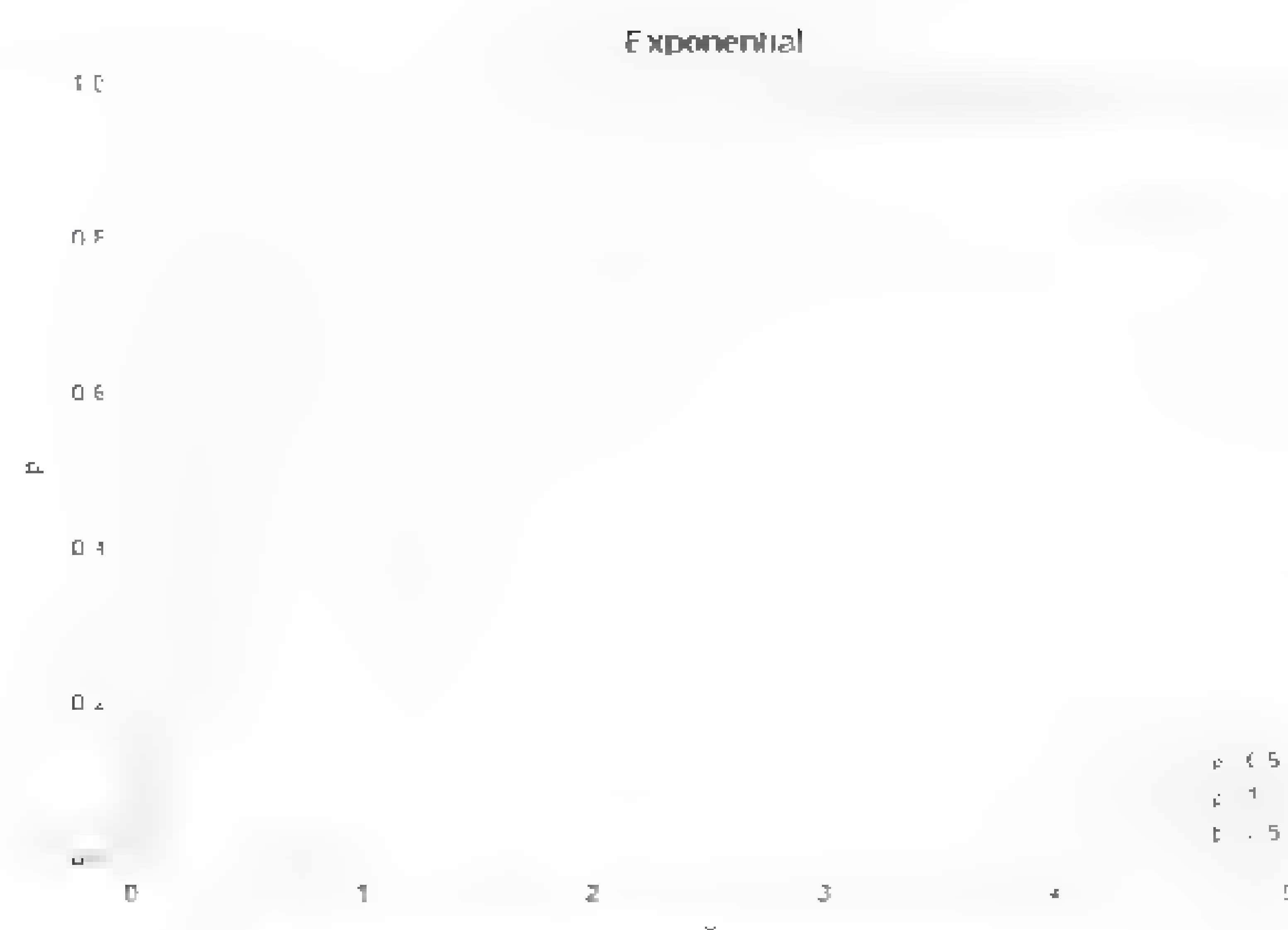


图 20-37　指数分布的累积分布曲线

指数分布可从泊松分布推导得出。假设下一个事件发生需要时间间隔为 t，则等同于在 t 之内没有任何事件发生，其概率记为

$$P(x>t)=P(N(t)=0)=\frac{(\lambda t)^0 e^{-\lambda t}}{0!}=e^{-\lambda t}$$

则反过来，事件在时间 t 内发生的概率就是 1 减去上面的值：

$$P(X\leqslant t)=1-P(X>t)=1-e^{-\lambda t}$$

假设某医院平均每 60min 出生 3 个婴儿，则可以计算接下来的 15min（等于 $\frac{1}{4}\times 60\text{min}$，因此 $t=1/4=0.25$）内有婴儿出生的概率约为 53%。其计算方法为

$$P(X\leqslant 0.25)=1-e^{-3\times 0.25}\approx 0.5276$$

而接下来的 15min 到 30min 内有婴儿出生的概率约为 25%。

$$P(0.25\leqslant X\leqslant 0.5)=P(X\leqslant 0.5)-P(X\leqslant 0.25)$$
$$=(1-e^{-3\times 0.5})-(1-e^{-3\times 0.25})=e^{-0.75}-e^{-1.5}\approx 0.2492$$

该计算过程也可使用 SAS 程序 20-35 求解如下

程序20-35　计算泊松事件在接下来的特定时段内发生的概率

```
data _null_;
  lamda=3; /*每60分钟内发生3次试验成功*/
  t0=60;

  n=0;

  tx=15;
  x=tx/t0;
  m=lamda * x;
  p15=pdf('POISSON', n, m);
  q15=1- p15; /*15 分钟内有婴儿出生的概率*/
  put q15= ;

  p15_x=pdf('Exponential', m ); /*Possion(n,m)用Exponential(m)替代*/
  q15_x=1- p15_x;
  put q15_x= ;

  tx=30;
  x=tx/t0;
  m=lamda * x;
```

```
  p30=pdf('Exponential', m );
  q30=1- p30;/*30 分钟内有婴儿出生的概率*/
  put q30= ;

  p_15_30=q30-q15;/*15 - 30 分钟内有婴儿出生的概率*/
  put p_15_30=;

run;
```

系统输出：

```
q15=0.5276334473
q15_x=0.5276334473
q30=0.7768698399
p_15_30=0.2492363926
```

20.3.4 卡方分布

如果独立随机变量 Y_i 服从均值为 0，方差为 1 的标准正态分布，则 r 个变量 Y_i 的平方和 X^2 所服从的统计分布，称为自由度为 r 的卡方分布 (Chi-square Distribution)，通常记为 $\chi^2(r)$。

$$X^2=\sum_{i=1}^{r}Y_i^2$$

卡方分布是假设检验开山鼻祖统计学家卡尔•皮尔逊的发明。自由度为 r 的卡方分布，其概率密度函数和累积分布函数为

$$P_r(x)=\frac{x^{\frac{r}{2}-1}\mathrm{e}^{-x/2}}{\Gamma(\frac{1}{2}r)\,2^{r/2}}$$

$$D_r(X^2)=\frac{\gamma(\frac{1}{2}r,\ \frac{1}{2}X^2)}{\Gamma(\frac{1}{2}r)}$$

其中 x 属于 0 到正无穷，$\Gamma\left(\frac{1}{2}r\right)$ 为伽马函数，而 $\gamma\left(\frac{1}{2}r,\frac{1}{2}X^2\right)$ 为不完全伽马函数。卡方分布和伽马分布有密切关系，卡方分布 $\chi^2(r)$ 等价于 $\alpha=r/2$，$\beta=1/2$ 的伽马分布 Gamma(α,β)。

卡方分布总体均值、方法、偏度系数和峰度系数为

$$\mu=r$$
$$\sigma^2=2r$$
$$\gamma_1=2\sqrt{2/r}$$
$$\gamma_2=\frac{12}{r}$$

k 个独立服从不同自由度的卡方分布的变量 X_i^2，它们的和 $\sum_{j=1}^{k}X_j^2$ 仍然服从自由度为所有自由度之和的卡方分布，即独立卡方分布具有可加性：

$$\sum_{j=1}^{k}X_j^2\sim\chi^2\left(r=\sum_{j=1}^{k}r_j\right)$$

在统计学中对一个总体的方差或标准差进行检验时，会使用特定卡方统计量进行卡方检验，其原理就是该特定统计量的分布服从卡方分布。本书附录 5 列出了 χ^2 分布的临

界值表，可用于卡方检验。

在 SAS 里绘制自由度 r=1, 2, 3, 4, 5 的卡方分布的概率密度曲线和累积分布曲线如程序 20-36 和程序 20-37 所示，输出结果如图 20-38 和图 20-39 所示。

程序20-36　卡方分布的概率密度曲线

```
data sample;
  do x=0 to 8 by 0.01 ;
    p=pdf("CHISQUARE", x,  1);
    p2=pdf("CHISQUARE", x, 2);
    p3=pdf("CHISQUARE", x, 3);
    p4=pdf("CHISQUARE", x, 4);
    p5=pdf("CHISQUARE", x, 5);
    output;
  end;
run;
proc sgplot;
  title "CHISQUARE";
  series x=x y=p /  legendlabel="v=1" lineattrs=(pattern=solid color=red);
  series x=x y=p2 /  legendlabel="v=2" lineattrs=(pattern=dash color=green);
  series x=x y=p3 /  legendlabel="v=3" lineattrs=(pattern=dot color=blue); ;
  series x=x y=p4 /  legendlabel="v=4" lineattrs=(pattern=dashdashdot
    color=red); ;
  series x=x y=p5 /  legendlabel="v=5" lineattrs=(pattern=dashdashdot
    color=green); ;

  keylegend / location=inside position=topright across=1;
  yaxis max=1.0;
run;
```



图 20-38　卡方分布的概率密度曲线

程序20-37　卡方分布的累积分布曲线

```
data sample;
  do x=0 to 8 by 0.01 ;
    p=cdf("CHISQUARE", x,   1);
    p2=cdf("CHISQUARE", x,  2);
    p3=cdf("CHISQUARE", x,  3);
    p4=cdf("CHISQUARE", x,  4);
    p5=cdf("CHISQUARE", x, 5);
```

```
    output;
  end;
run;
proc sgplot;
  title "CHISQUARE";
  series x=x y=p / legendlabel="v=1"
    lineattrs=(pattern=solid color=red);
  series x=x y=p2 / legendlabel="v=2"
    lineattrs=(pattern=dash color=green);
  series x=x y=p3 / legendlabel="v=3"
    lineattrs=(pattern=dot color=blue); ;
  series x=x y=p4 / legendlabel="v=4"
    lineattrs=(pattern=dashdashdot color=red); ;
  series x=x y=p5 / legendlabel="v=5"
    lineattrs=(pattern=dashdashdot color=green); ;

  keylegend / location=inside position=bottomright across=1;
  yaxis max=1.0;
run;
```



图 20-39　卡方分布的累积分布曲线

20.3.5　学生*t*-分布

t 统计量是 1908 年统计学家威廉・哥赛特发表在《生物统计》期刊的论文《均值的或然误差》中首先提出的。该论文开创了小样本统计理论的先河并奠定了抽样分布的理论基础，被誉为统计推断理论发展史上的里程碑。在统计学上要从样本推断总体，样本必须是随机抽样所得的随机样本。由于他曾师从统计学家卡尔・皮尔逊，因此威廉・哥赛特一直用笔名“学生氏”发表论文，学生 *t* 分布 (Student’s *t* Distribution) 就是 *t* 统计量所服从的分布形式（见图 20-40）。

对于 n 个服从正态分布 $N(\mu, \sigma^2)$ 的独立观测 X_i，样本统计量 t 的定义如下：

$$t=\frac{\bar{x}-\mu}{s/\sqrt{n}}$$

其中，μ 为总体均值；$\bar{x}$为样本均值；s 为样本标准差。则随机变量 t 服从自由度为 $n-1$ 的学生 t- 分布，记为 $t(n-1)$。一般地，统计量 t 可用来对总体参数的单个假设进行检验，它等于（估计值－假设值）/ 标准误。具体到上面公式中样本均值 $\bar{x}$ 就是对总体均值的估计值，而 μ 就是总体均值的假设值，分母 $s/\sqrt{n}$ 为标准误差。

随机变量 t 是在不知道总体方差 σ 的情况下，几乎是对总体均值的最佳估计。也就是说随机抽样所得的小样本的平均值服从学生 t- 分布。学生 t- 分布只有一个自由度参数，其概率密度函数为

$$f_r(t)=\frac{\Gamma\left[\frac{1}{2}(r+1)\right]}{\sqrt{r\pi}\,\Gamma\left(\frac{1}{2}r\right)\left(1+\frac{t^2}{r}\right)^{(r+1)/2}}$$

其中 $r=n-1$ 为样本自由度，统计量 t 介于负无穷到正无穷之间。$\Gamma(z)$ 为伽马函数 $\Gamma(z)=\int_0^{\infty}\frac{t^{z-1}}{e^t}\mathrm{d}t$，当 z 为正整数时，$\Gamma(z)=(z-1)!$。

学生 t- 分布的总体均值、方差、偏度和峰度为

$$\mu=0$$
$$\sigma^2=\frac{r}{r-2}$$
$$\gamma_1=0$$
$$\gamma_2=\frac{6}{r-4}$$

学生 t- 分布与标准正态分布的均值都是 0 且分布曲线都是关于 0 左右对称，曲线下方的面积为 1.0。但标准正态分布曲线是与自由度无关的唯一分布曲线，而学生 t- 分布对于每一个自由度都有一条特有的分布曲线，并不断向标准正态分布曲线逼近。学生 t- 分布的概率密度曲线和累积分布曲线如图 20-40 所示，更多分布曲线参考本节后面的样例代码。



图 20-40　学生 t- 分布的概率密度曲线和累积分布曲线

若随机变量 X 和 Y 互相独立，且随机变量 X 服从均值为 0，方差为 σ^2 的标准正态分布 $X\sim N(0,\sigma^2)$，而 Y^2/σ^2 服从自由度为 n 的卡方分布 $Y^2/\sigma^2\sim\chi^2(n)$，则如下随机变量：

$$t=\frac{X\sqrt{n}}{Y}$$

服从自由度为 n 的学生分布。该性质将标准正态分布，卡方分布和学生 t- 分布 3 种分布关联起来。学生 t- 分布与正态分布关系密切，且随着样本容量 n 的增加，学生 t- 分布逼近于正态分布。t 统计量经常用于总体方差 σ^2 未知时，利用小样本对总体均值进行估计。该统计量在一个总体均值以及两个总体均值之差的参数检验中也经常使用到。

● 学生 z-分布

除了学生 t- 分布，还有一个与之关系密切的分布叫学生 z- 分布，其定义为

$$z=\frac{\overline{x}-\mu}{s}$$

其中 $\overline{x}$ 为样本均值，而 μ 为总体均值。则学生 z- 分布的概率密度函数和累积分布函数为

$$f_n(z)=\frac{\Gamma\left(\frac{n}{2}\right)}{\sqrt{\pi}\Gamma\left(\frac{n-1}{2}\right)}(1+z^2)^{-\frac{n}{2}}$$

$$D_n(z)=\begin{cases}d_n(z), & z\leqslant 0\\ 1-d_n(z), & z>0\end{cases}$$

其中

$$d_n(z)=\frac{|z|^{1-n}\Gamma\left(\frac{1}{2}n\right)\ {}_2F_1\left(\frac{1}{2}(n-1),\frac{1}{2}n;\frac{1}{2}(n+1);-z^{-2}\right)}{2\sqrt{\pi}\Gamma\left(\frac{1}{2}(n+1)\right)}$$

学生 z- 分布的总体均值、方差、偏度和峰度系数为

$$\mu=0$$

$$\sigma^2=\frac{1}{n-3}$$

$$\gamma_1=0$$

$$\gamma_2=\frac{6}{n-5}$$

学生 z- 分布和学生 t- 分布之间可相互转换，其变换公式为

$$z=\frac{\overline{x}-\mu}{s}$$

$$t=z\sqrt{n-1}$$

SAS 程序 20-38 和程序 20-39 可生成自由度为 1，2，4 以及正无穷的学生 t- 分布图，其概率密度曲线和累积分布曲线如图 20-41 和图 20-42 所示。

程序20-38 学生t-分布图的概率密度曲线

```
data sample;
  do x=-5 to 5 by 0.01 ;
    p=pdf('T', x,   1);
    p2=pdf('T', x,   2);
    p3=pdf('T', x,   5);
    p4=pdf('T', x,  constant("BIG"));
    output;
  end;
run;
proc sgplot;
  title "T";
  series x=x y=p /  legendlabel="v=1" lineattrs=(pattern=solid color=red);
```

```
  series x=x y=p2 /  legendlabel="v=2" lineattrs=(pattern=dash color=green);
  series x=x y=p3 /  legendlabel="v=5" lineattrs=(pattern=dot color=blue); ;
  series x=x y=p4 /  legendlabel="v=∞" lineattrs=(pattern=dashdashdot
    color=red); ;

  keylegend / location=inside position=topright across=1;
  yaxis max=0.4;
run;
```

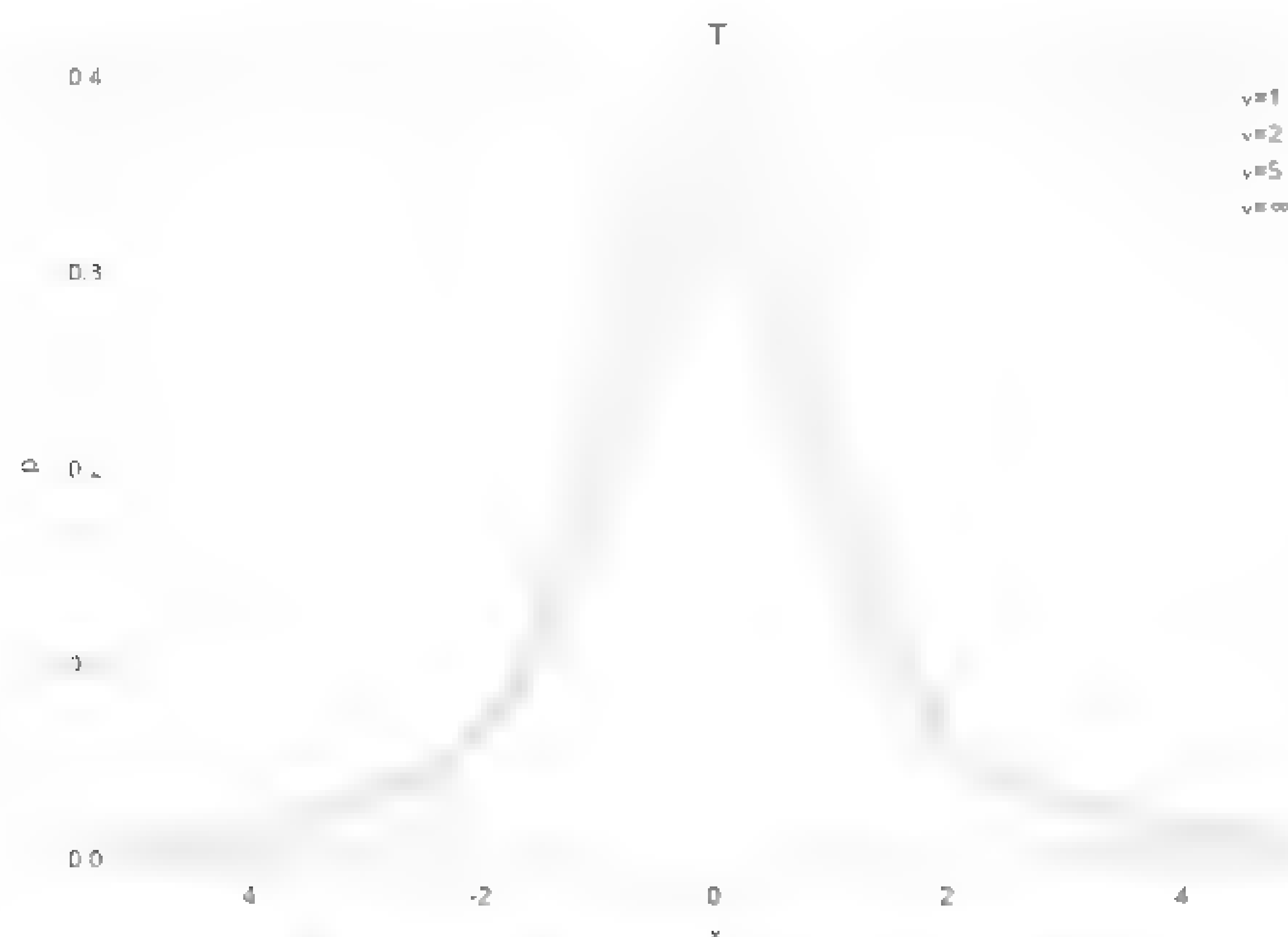


图 20-41　学生 *t*- 分布的概率密度曲线

程序20-39　学生t-分布的概率密度曲线

```
data sample;
  do x=-5 to 5 by 0.01 ;
    p=cdf('T', x,   1);
    p2=cdf('T', x,  2);
    p3=cdf('T', x,  5);
    p4=cdf('T', x,  32767 );
    output;
  end;
run;
proc sgplot;
  title "T";
  series x=x y=p /  legendlabel="v=1" lineattrs=(pattern=solid color=red);
  series x=x y=p2 /  legendlabel="v=2" lineattrs=(pattern=dash color=green);
  series x=x y=p3 /  legendlabel="v=5" lineattrs=(pattern=dot color=blue); ;
  series x=x y=p4 /  legendlabel="v=∞" lineattrs=(pattern=dashdashdot
    color=red); ;

  keylegend / location=inside position=bottomright across=1;
run;
```

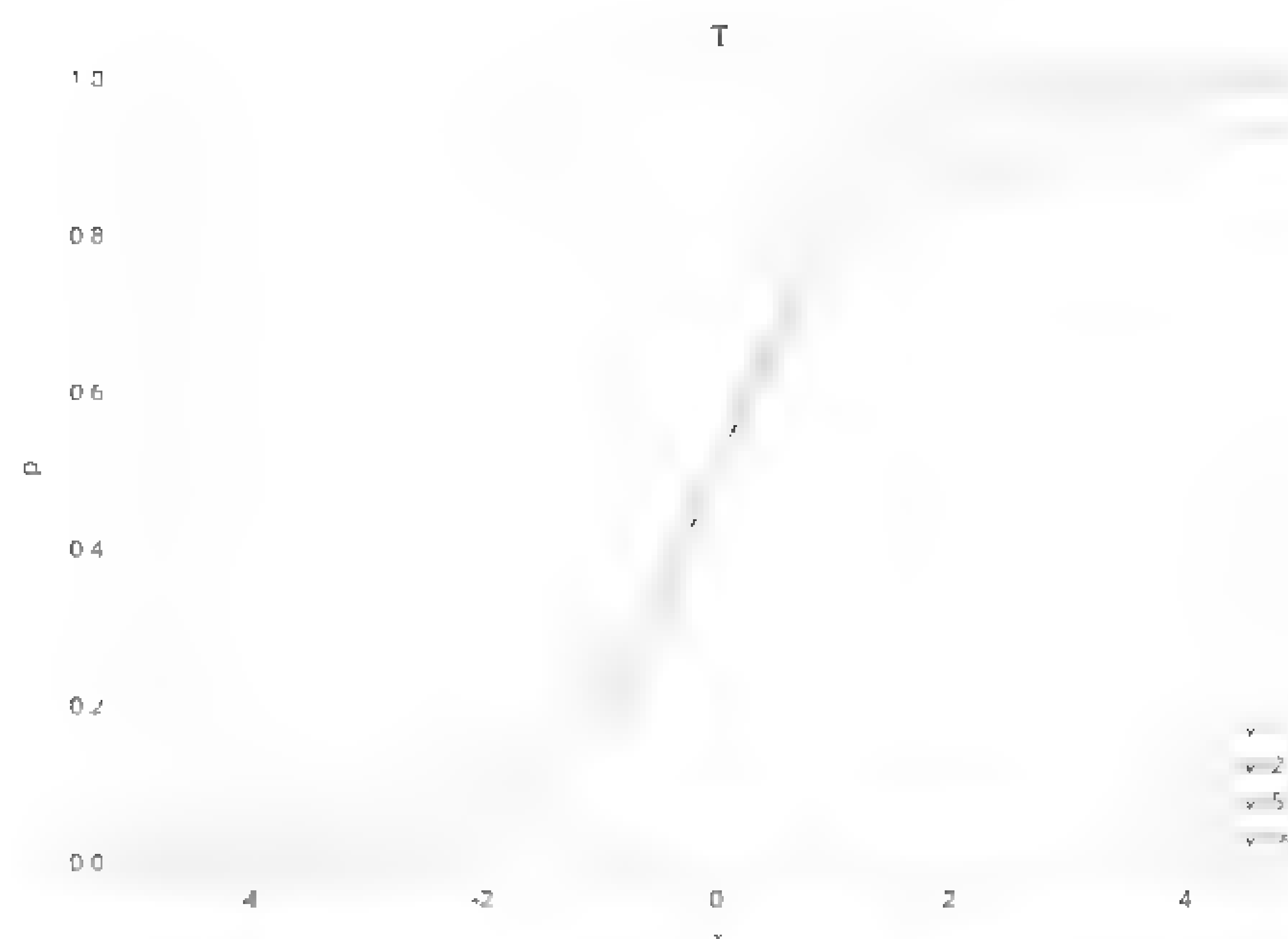


图 20-42　学生 t- 分布的累积分布曲线

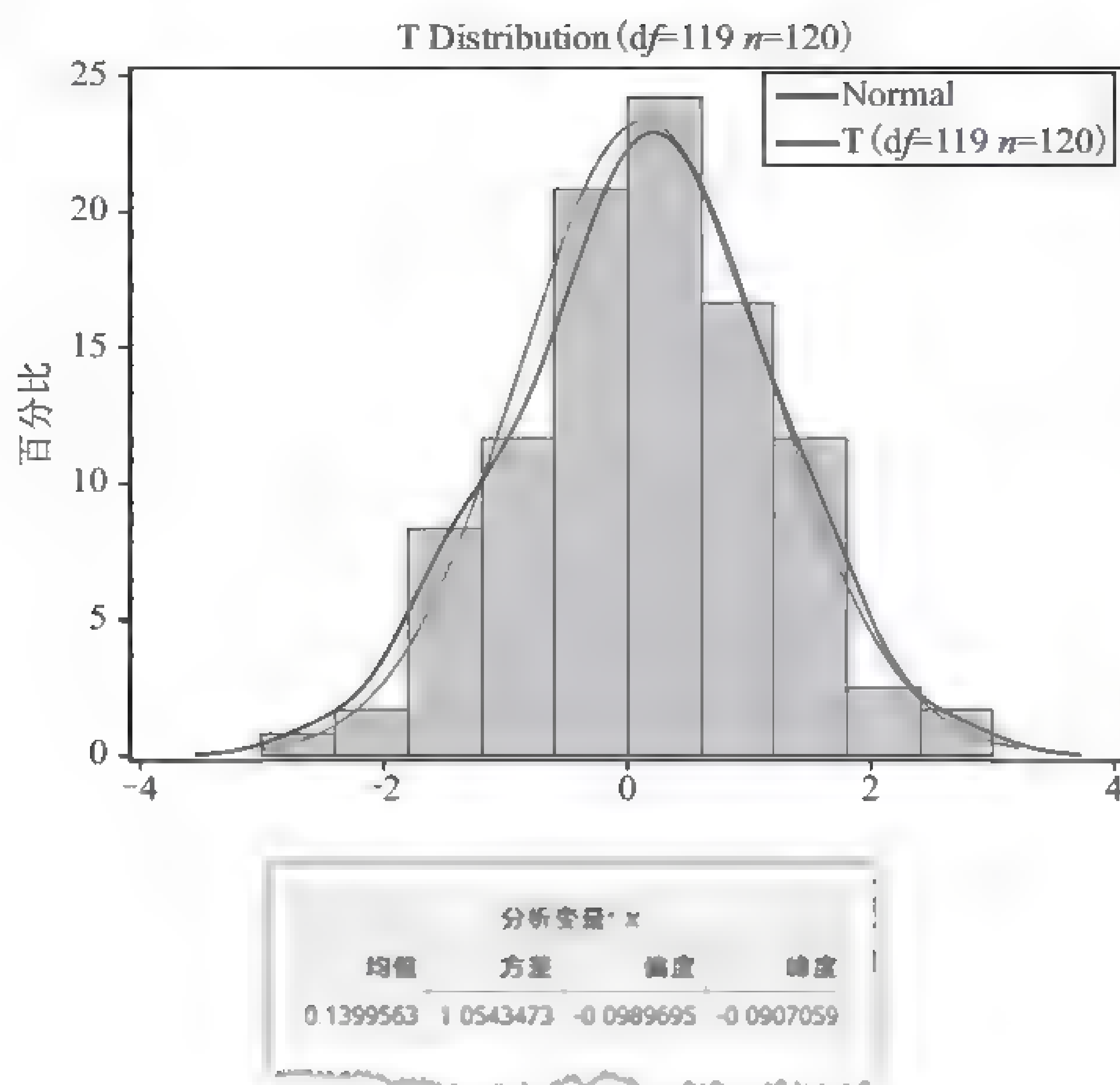
学生 *t*- 分布在样本容量 *n* 大于等于 120 时，即自由度 df=119 时，*t*- 分布与正态分布已经非常逼近（见程序 20-40 和图 20-43）。

程序20-40　学生t-分布在样本容量较大时逼近正态分布

```
data sample;
  n=200;
  do i = 1 to n;
    df=n-1;
    x=RAND('T', df); /*df > 0 */
  output;
  end;
  keep x;
run;
proc sgplot data=sample;
  title 'T Distribution (df=4 n=5) ';
  histogram x      ;

  density x / type=normal legendlabel='Normal'
    lineattrs=(pattern=solid color=red);
  density x / type=kernel legendlabel='T (df=4 n=5)'
    lineattrs=(pattern=solid color=blue);
  keylegend / location=inside position=topright across=1;
  xaxis display=(nolabel);
run;

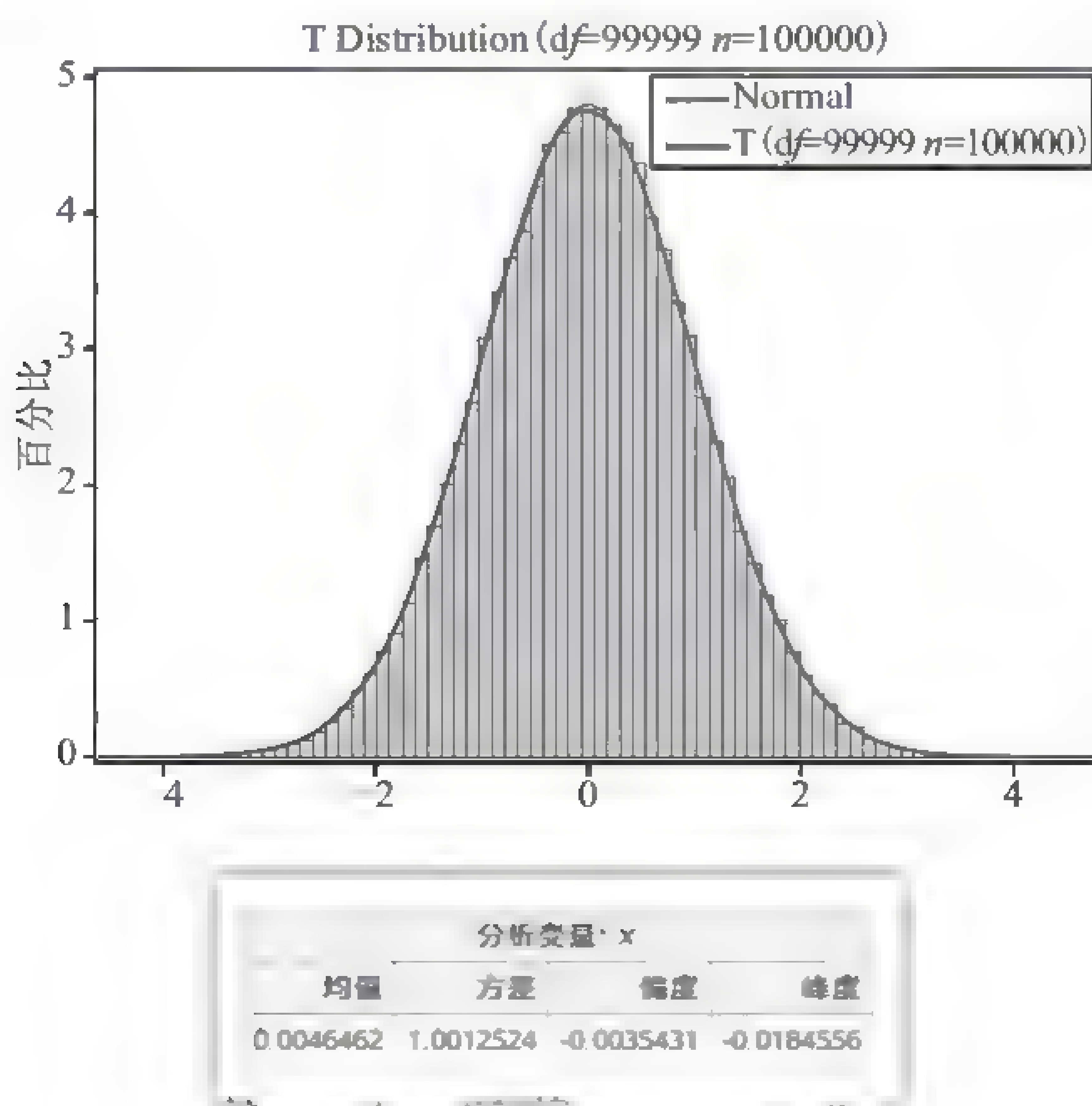
proc means data=sample mean var skewness kurtosis; run;
```



分析变量：x			
均值	方差	偏度	峰度
0.1399563	1.0543473	-0.0989695	-0.0907059

图 20-43　学生 *t*- 分布与正态分布 (df=119)

学生 *t*- 分布在 *n* 值很大时接近于标准正态分布，比如 *n*=100000，df=99999 *t*- 分布如图 20-44 所示。



分析变量：x			
均值	方差	偏度	峰度
0.0046462	1.0012524	-0.0035431	-0.0184556

图 20-44　学生 *t*- 分布逼近正态分布

检查其均值和方差果然为0和1，而且偏度和峰度也近于0。也就是说，*t*-分布有个重要的性质：在样本容量 n 值很大时它逼近于标准正态分布 $N(0, 1)$。

20.3.6　*F*分布

*F*分布（Fisher's F Distribution）是以推断统计创立者R. A. 费歇的名字命名的统计分布，主要用于检测两个样本是否具有相同方差时所发明的连续型统计分布，通常记为 $F(n, m)$。

假定两个随机变量 X_n^2 和 X_m^2 相互独立，且分别服从自由度为 n 和 m 的卡方分布，则定义如下统计量 F_{nm} 为

$$F_{nm} = \frac{X_n^{\ 2} / n}{X_m^{\ 2} / m}$$

则该统计量也是一个随机变量，其中有 F 大于等于0到正无穷。F 分布的概率密度函数和累积分布函数为

$$f_{nm}(x) = \frac{m^{\frac{m}{2}} n^{\frac{n}{2}} x^{\frac{n}{2}-1}}{(m+nx)^{\frac{n+m}{2}} B\left(\frac{1}{2}n, \frac{1}{2}m\right)}$$

$$F_{nm}(x) = I\left(\frac{nx}{m+nx}; \frac{1}{2}n, \frac{1}{2}m\right)$$

其中 $B(a, b)$ 为贝塔函数；$I(x; a, b)$ 为归一化贝塔函数。F 分布的总体均值、方差、偏度系数和峰度系数为

$$\mu = \frac{m}{m-2}$$

$$\sigma^2 = \frac{2m^2(m+n-2)}{n(m-2)^2(m-4)}$$

$$\lambda_1 = \frac{2(m+2n-2)}{m-6}\sqrt{\frac{2(m-4)}{n(m+n-2)}}$$

$$\lambda_2 = \frac{12(-16+20m-8m^2+m^3+44n-32mm+5m^2n-22n^2+5mn^2)}{n(m-6)(m-8)(n+m-2)}$$

F 统计量在对两个样本的总体方差之比进行估计和检验时具有重要作用。F 检验在单因素和双因素方差分析中扮演着极其重要的角色，其原理就是组间变异和组内变异的比这一统计量，服从特定自由度的 F 分布。

在SAS中生成 F 分布的概率密度曲线程序如程序20-41所示，输出结果如图20-45所示。读者可以绘制其累积分布曲线如图20-46所示。

程序20-41　*F* 分布的概率密度曲线和累积分布曲线

```
data sample;
  do x=0 to 5 by 0.01 ;
    p=pdf("F", x,   1,1);
    p2=pdf("F", x,  2,1);
    p3=pdf("F", x,  5,2);
    p4=pdf("F", x,  100,1);
    p5=pdf("F", x, 100,100);
    output;
  end;
run;
proc sgplot;
  title "F";
  series x=x y=p /  legendlabel=" d1=1 d2=1"
    lineattrs=(pattern=solid color=red);
  series x=x y=p2 /  legendlabel=" d1=2 d2=1"
    lineattrs=(pattern=dash color=green);
  series x=x y=p3 /  legendlabel=" d1=5 d2=2"
    lineattrs=(pattern=dot color=blue); ;
  series x=x y=p4 /  legendlabel=" d1=100 d2=1"
    lineattrs=(pattern=dashdashdot color=red); ;
  series x=x y=p5 /  legendlabel=" d1=100 d2=100"
    lineattrs=(pattern=dashdotdot color=green); ;

  keylegend / location=inside position=topright across=1;
  yaxis max=3;
run;
```

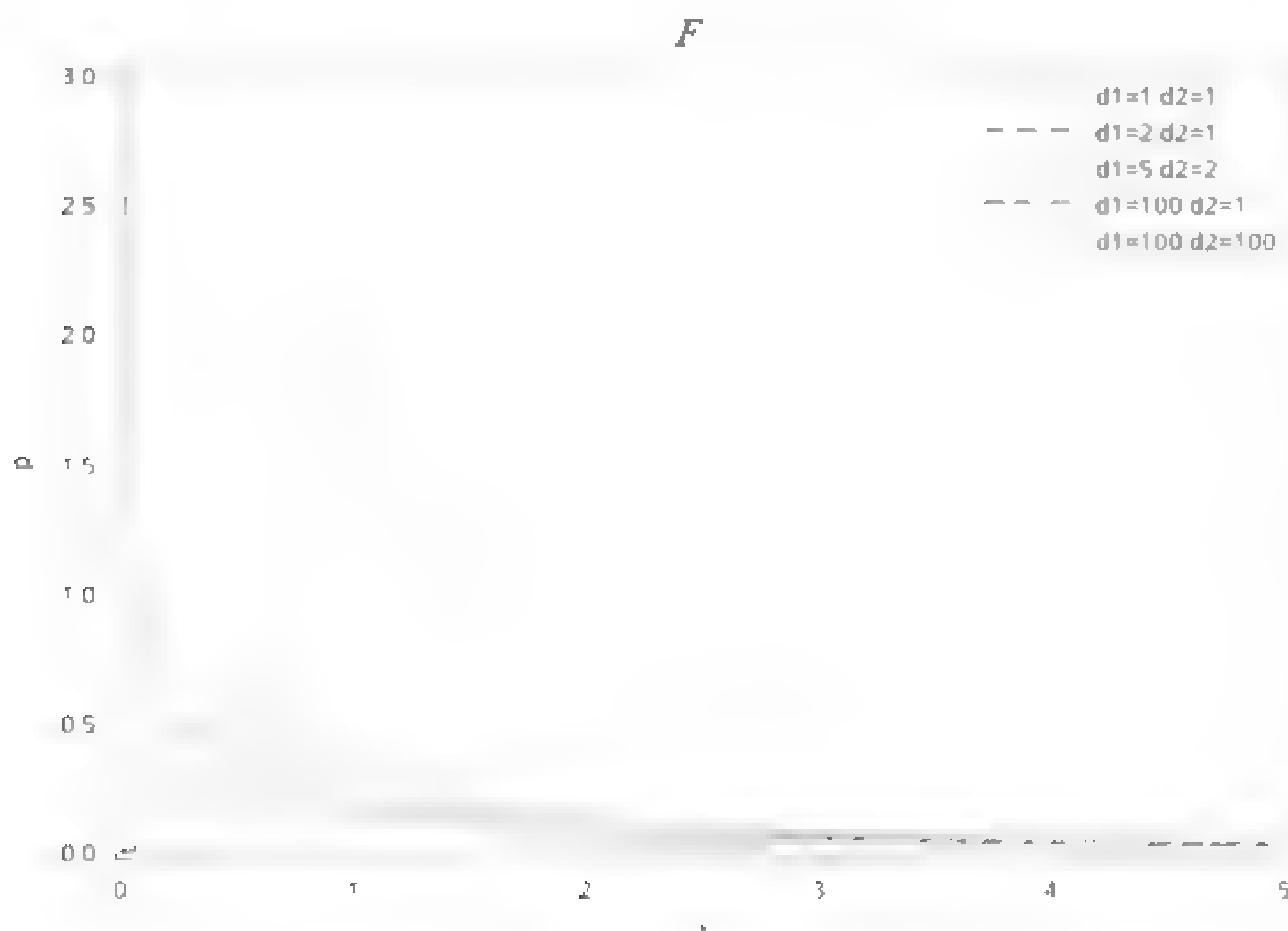


图 20-45　*F* 分布的概率密度曲线

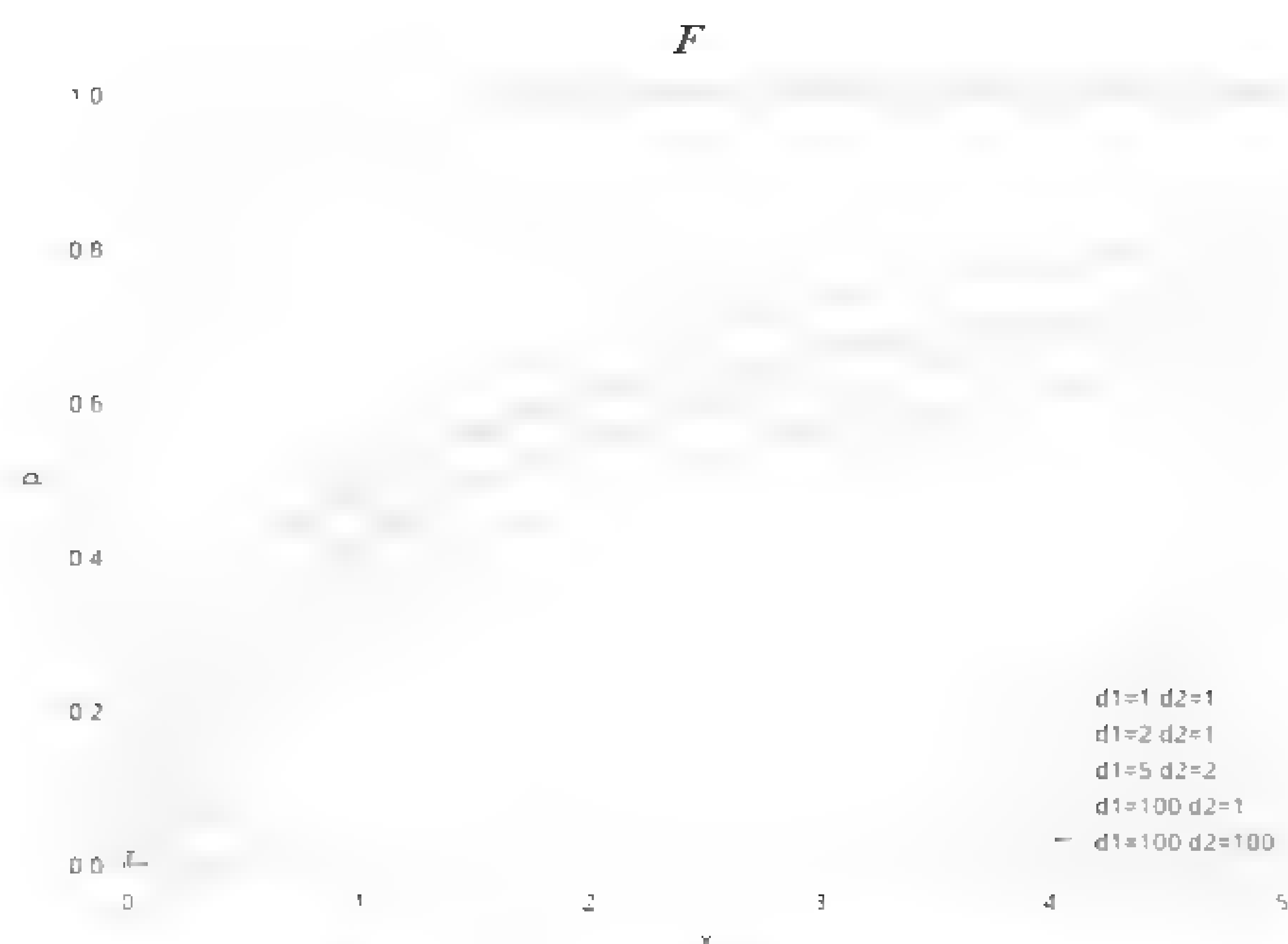


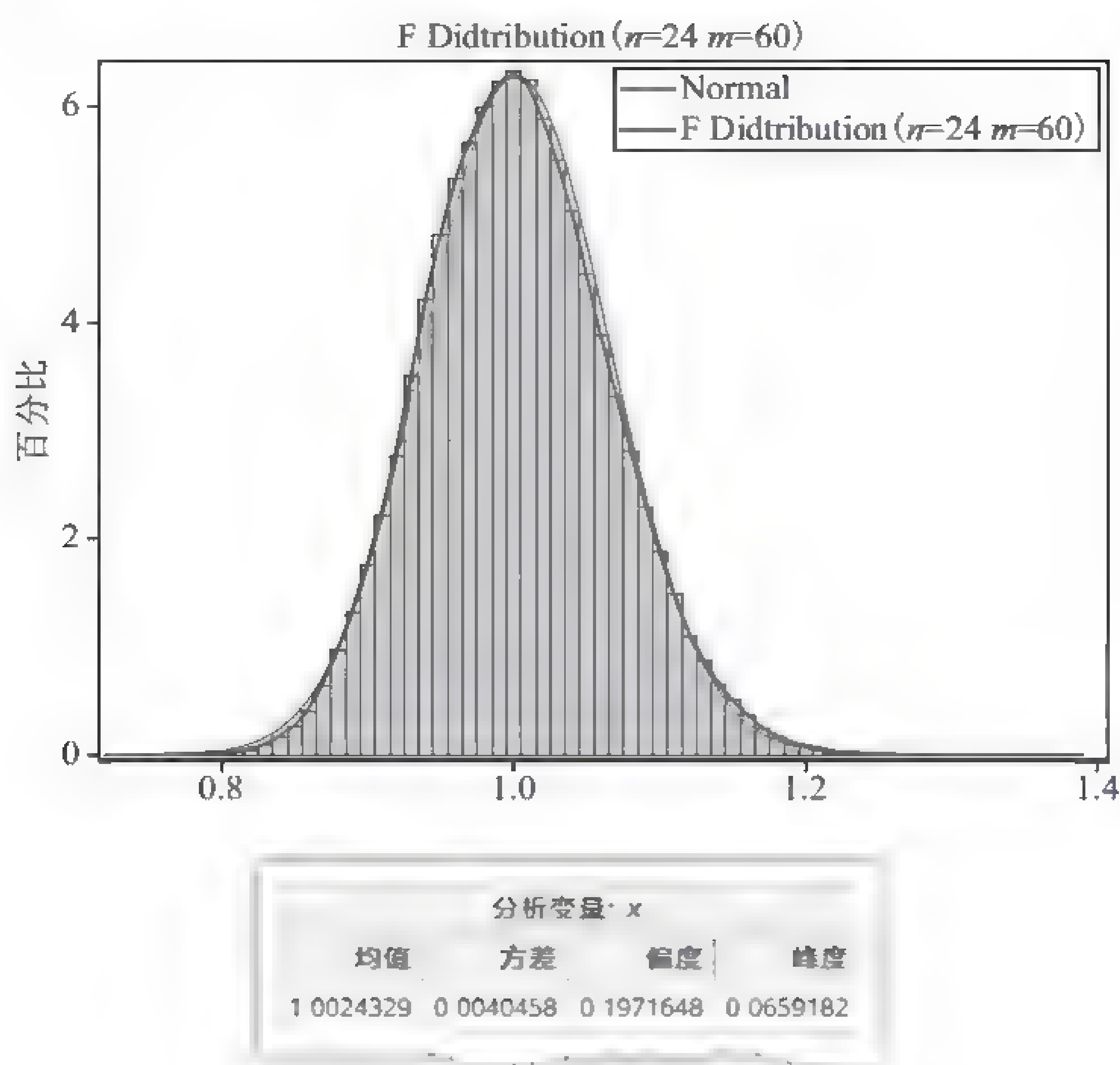
图 20-46 *F* 分布的累积分布曲线

F 分布在 *n* 和 *m* 在足够大时（如 $n=m=1000$），*F* 分布无限逼近均值为 1，方差为 0 的正态分布（见程序 20-42 和图 20-47）。需要特别注意，它逼近的分布不是均值为 0，方差为 1 标准正态分布，而是均值为 1 方差为 0 的正态分布。

程序20-42 *F* 分布在*n*和*m*在足够大时逼近均值为1，方差为0的正态分布

```
data sample;
  do i = 1 to 100000 ;
    n=24; m=60;
    x=RAND('F', n, m) ;/*x > 0; n>0; m > 0  */
    output;
  end;
  keep x ;
run;
proc sgplot data=sample;
  title 'F Distribution (n=24 m=60) ';
  histogram x     ;

  density x / type=normal legendlabel='Normal'
    lineattrs=(pattern=solid color=red);
  density x / type=kernel legendlabel='F Distribution (n=24 m=60) '
    lineattrs=(pattern=solid color=blue);
  keylegend / location=inside position=topright across=1;
  xaxis display=(nolabel);
run;
proc means data sample mean var skewness kurtosis; run;
```



分析变量：x			
均值	方差	偏度	峰度
1.0024329	0.0040458	0.1971648	0.0659182

图 20-47　F 分布在 n, m 很大时逼近正态分布

20.3.7　柯西分布

柯西分布 (Cauchy Distribution) 也称洛仑兹分布，是一个数学期望不存在的连续型分布。它可用于描述共振行为，也可形象地用一个悬挂在高度为 b 的 y 轴上的锚点，以随机角度 θ 投射在水平轴上截取的长度 x 的分布来表示（见图 20-48）。另外，由于服从正态分布的两个随机变量 X 与 Y 的比服从柯西分布，因此柯西分布也被视作正态分布的姊妹分布。

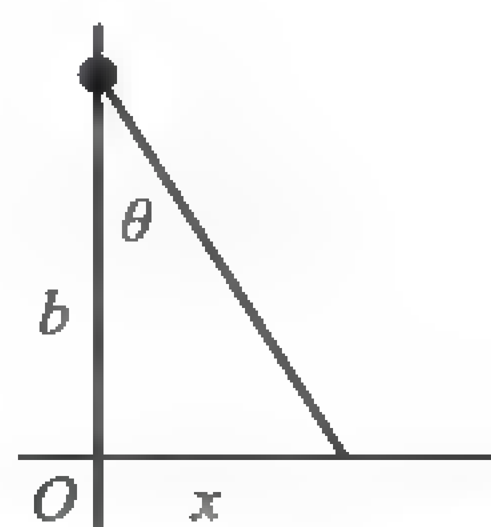


图 20-48　柯西分布原理

其概率密度函数和累积分布函数为

$$P(x)=\frac{1}{\pi}\frac{b}{(x-m)^2+b^2}$$

$$D(x)=\frac{1}{2}+\frac{1}{\pi}tan^{-1}\left(\frac{x-m}{b}\right)$$

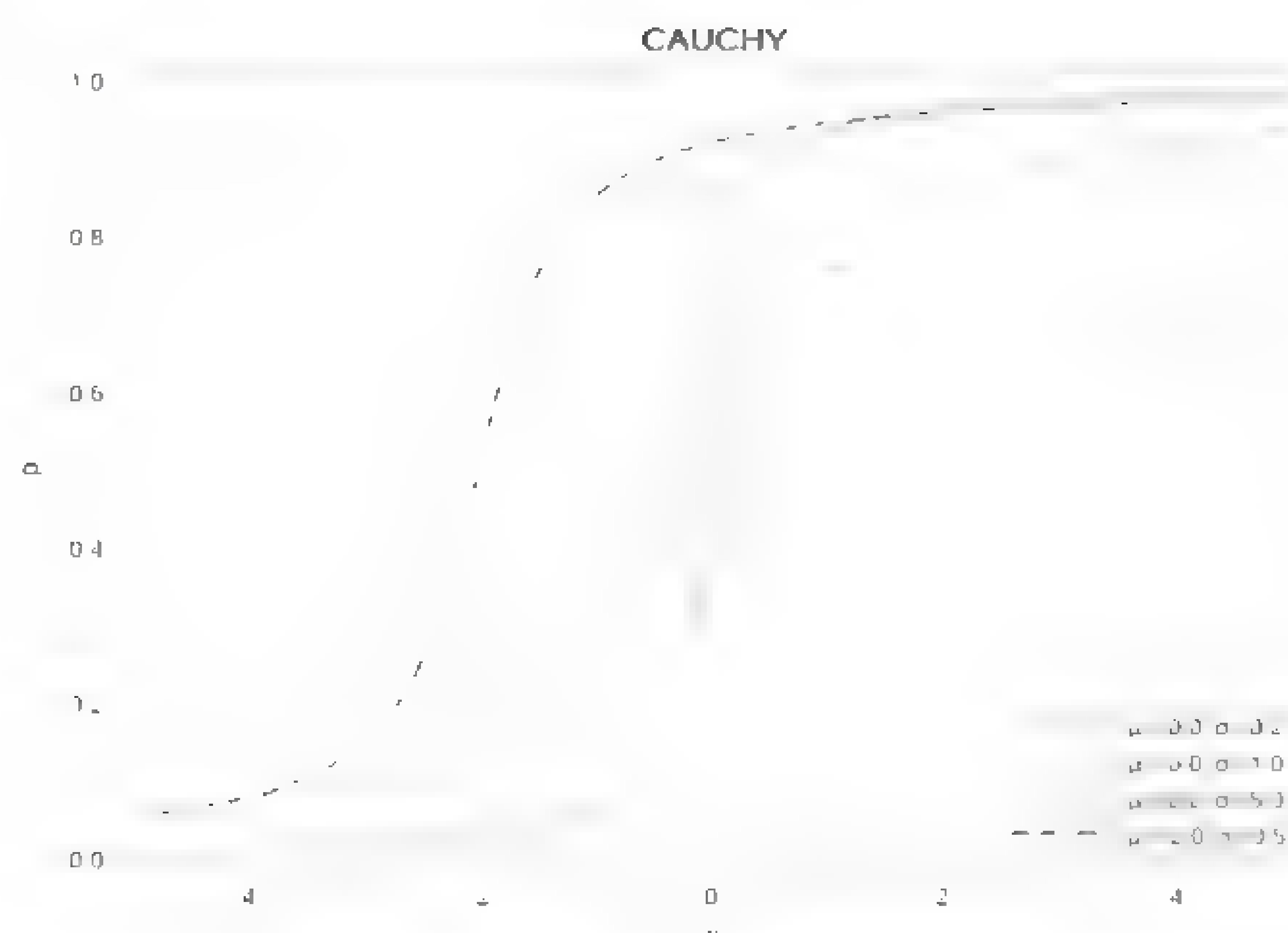


图 20-51　柯西分布的累积分布曲线

20.3.8　贝塔分布

贝塔分布（Beta Distribution）在贝叶斯分析中被当作是二项式比例的先验分布，它有两个自由参数：α 和 β，通常记为 Beta(α, β)。图 20-52 展示了 α=1，β=0.25~3.0 的概率密度曲线和分布曲线的变化。

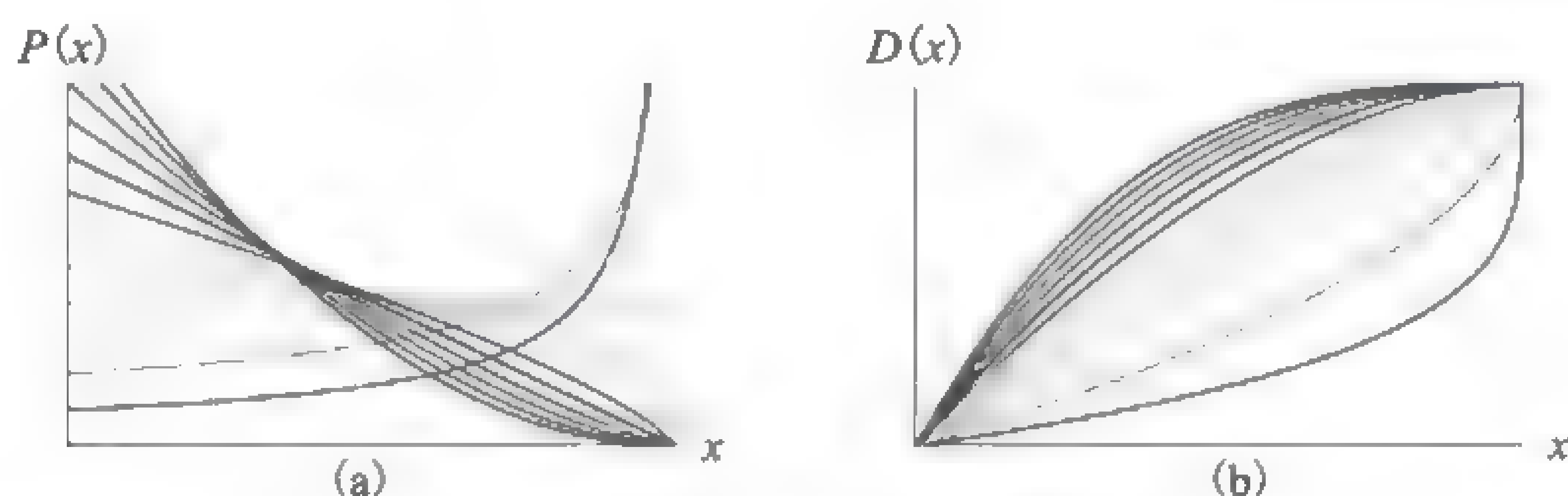


图 20-52　贝塔分布的概率密度曲线和累积分布曲线

贝塔分布的概率密度函数和累积分布函数为

$$P(x)=\frac{(1-x)^{\beta-1}x^{\alpha-1}}{B(\alpha,\beta)}$$

$$D(x)=I(x;a,b)$$

其中 $\alpha,\beta>0$，$B(\alpha,\beta)$ 为贝塔函数，其中贝塔函数与伽马函数关系紧密，有 $B(\alpha,\beta)=\frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$ 而 $I(x;a,b)$ 为归一化贝塔函数。

贝塔分布的总体均值，方差、偏度和峰度为

$$\mu = \frac{\alpha}{\alpha+\beta}$$

$$\sigma^2 = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$$

$$\gamma_1 = \frac{2(\beta-\alpha)\sqrt{1+\alpha+\beta}}{\sqrt{\alpha\beta}(2+\alpha+\beta)}$$

$$\gamma_2 = \frac{6\left[\alpha^3+a^2(1-2\beta)+\beta^2(1+\beta)-2\alpha\beta(2+\beta)\right]}{\alpha\beta(\alpha+\beta+2)(\alpha+\beta+3)}$$

用 SAS 绘制 Beta 分布的概率密曲线见程序 20-44，其输出如图 20-53 所示。贝塔分布的累积分布曲线如图 20-54 所示。

程序20-44 贝塔分布的概率密度曲线

```
data sample;
  do x=0 to 1 by 0.01 ;
    p=pdf("BETA", x, 0.5, 0.5);
    p2=pdf("BETA", x,  5, 1);
    p3=pdf("BETA", x,  1, 3);
    p4=pdf("BETA", x,  2, 2);
    p5=pdf("BETA", x,  2, 5);
    output;
  end;
run;
proc sgplot;
  title "BETA";
  series x=x y=p /  legendlabel="α=0.5, β=0.5"
    lineattrs=(pattern=solid color=red);
  series x=x y=p2 /  legendlabel="α=5,   β=1 "
    lineattrs=(pattern=dash color=green);
  series x=x y=p3 /  legendlabel="α=1,    β=3"
    lineattrs=(pattern=dot color=blue); ;
  series x=x y=p4 /  legendlabel="α=2,   β=2"
    lineattrs=(pattern=dashdashdot color=red);
  series x=x y=p5 /  legendlabel="α=2,   β=5"
    lineattrs=(pattern=dashdotdot color=green);

  keylegend / location=inside position=topcenter across=1;
  yaxis max=5;
run;
```



图 20-53 贝塔分布的概率密度曲线



图 20-54　贝塔分布的累积分布曲线

当贝塔分布在 $\alpha=\beta=1$ 时，即 Beta(1, 1) 蜕变为连续均匀分布 Unif(0, 1)。可用 SAS 程序 20-45 来验证，输出结果如图 20-55 所示。

程序20-45　当a=b=1 时贝塔分布蜕变为最简单的均匀分布

```
data sample;
  do x=0 to 1.5 by 0.01 ;
    p=pdf("BETA", x,   1, 1);
    output;
  end;
run;
proc sgplot;
  title "BETA";
  series x=x y=p /  legendlabel="α=1, β=1"
    lineattrs=(pattern=solid color=red);
  keylegend / location=inside position=topright across=1;

run;
```

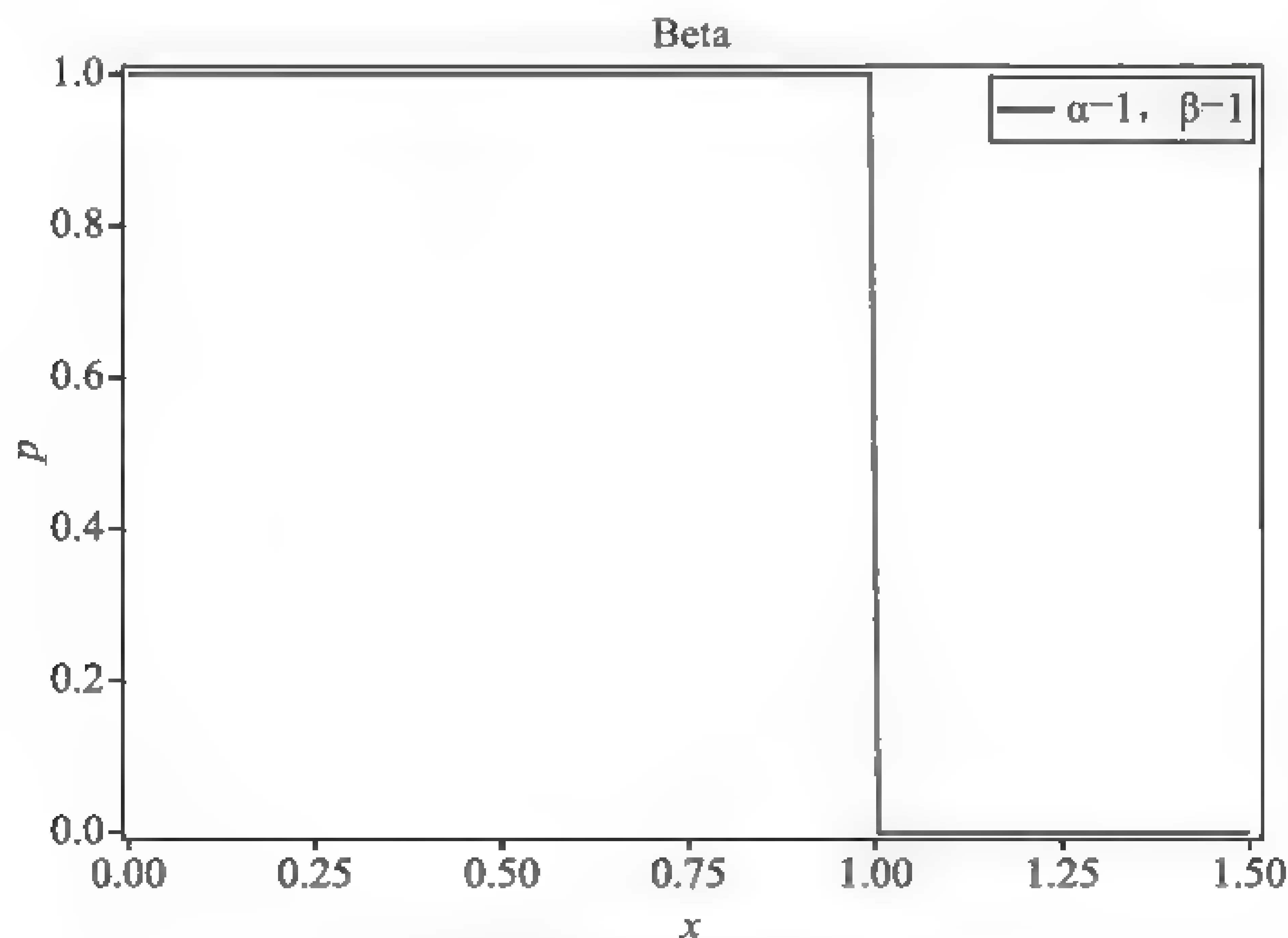


图 20-55　均匀分布与贝塔分布

20.3.9　伽马分布

伽马分布（Gamma Distribution）是从泊松分布的事件之间的等待时间过程中自然导出的统计分布，它也是与贝塔分布相关的一般统计分布。伽马分布有两个自由参数 α 和 β，通常记为 Gamma(α, β)。对于给定发生率 λ 的泊松分布，在第 h 次泊松事件发生之前的等待时间的分布函数 $D(x)$ 为

$$
\begin{aligned}
D(x) &= P(X \leqslant x) \\
&= 1 - P(X > x) \\
&= 1 - \sum_{k=0}^{h-1} \frac{(\lambda k)^k \mathrm{e}^{-\lambda x}}{k!} \\
&= 1 - \mathrm{e}^{-\lambda x} \sum_{k=0}^{h-1} \frac{(\lambda k)^k}{k!} \\
&= 1 - \frac{\Gamma(h, x\lambda)}{\Gamma(h)}
\end{aligned}
$$

其中 x 大于等于 0 到正无穷，$\Gamma(x)$ 为完全伽马函数，$\Gamma(a, x)$ 为不完全伽马函数。当 h 为整数时，该分布就是爱尔朗分布 (Erlang Distribution)。

对伽马分布的累积分布函数 $D(x)$ 求微分可得伽马分布的概率密度函数：

$$P(x) = D'(x) = \frac{\lambda(\lambda x)^{h-1}}{(h-1)!} \mathrm{e}^{-\lambda x}$$

假定 $\alpha=h$（α 不一定要求是整数），$\beta=1/\lambda$ 为泊松事件之间的时间，则上面的方程就演变为

$$P(x) = \frac{\beta^{\alpha}}{\Gamma(\alpha)} x^{\alpha-1} \mathrm{e}^{-\beta x}$$

伽马分布的 4 个核心统计量总体均值、方差、偏度系数和峰度系数为

$$\mu = \frac{\alpha}{\beta}$$

$$\sigma^2 = \frac{\alpha}{\beta^2}$$

$$\gamma_1 = \frac{2}{\sqrt{\alpha}}$$

$$\gamma_2 = \frac{6}{\alpha}$$

图 20-56 为 α 1, 1, 2 和 β 1, 2, 3 的概率密度函数和累积分布函数曲线。

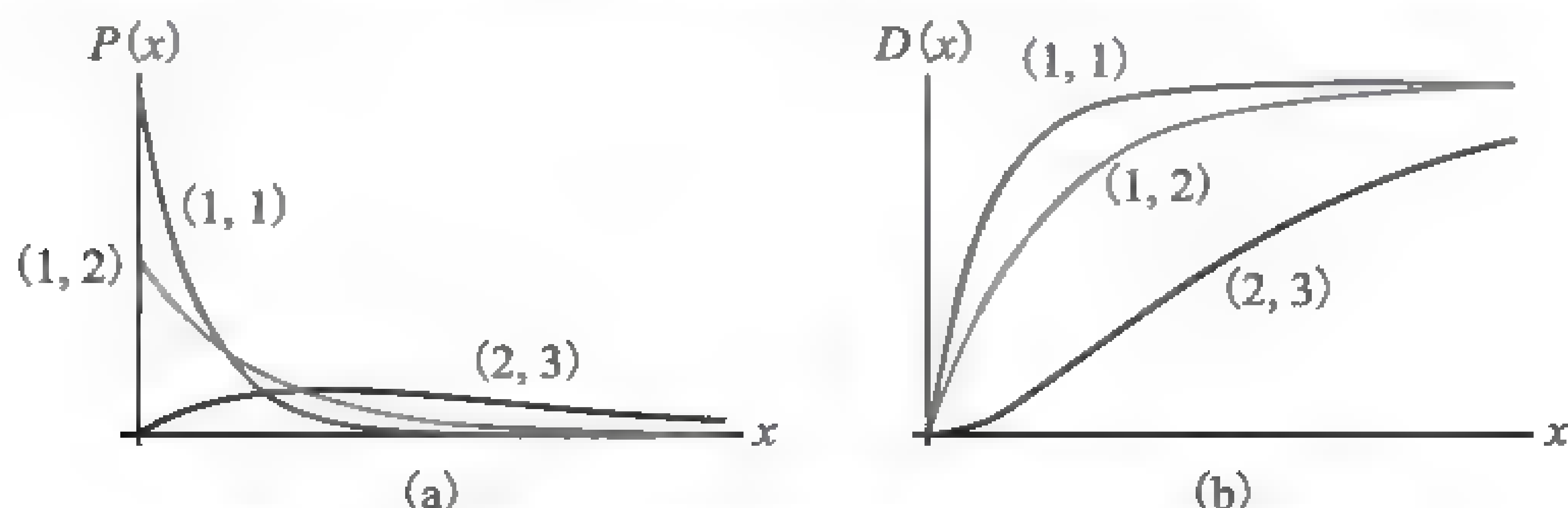


图 20-56　伽马分布的概率密度曲线与累积分布曲线

伽马分布具有如下一些性质值得特别关注：

(1) 若随机变量 X 服从 Gamma(α, β) 分布，则 X/β 服从 Gamma(α, 1) 分布。

(2) 伽马分布 Gamma(α, β) 本身服从 $\sum_{i=1}^{\alpha}\text{Exp}(\beta)$ 的指数分布。

(3) n 个服从伽马分布的独立随机变量 X_n，如果具有不同 α 值但有相同 β 值，则这些随机变量的和 $\sum_{i=1}^{n}X_i$ 一定服从 $\alpha=\sum_{i=1}^{n}\alpha_i$ 的伽马分布，即 $\text{Gamma}\left(\alpha=\sum_{i=1}^{n}\alpha_i, \beta\right)$。也就说相同 β 的伽马分布具有可加性。

(4) 2 个服从伽马分布的独立随机变量 X_1 和 X_2，如果它们具有不同的 α 值（α_1 和 α_2）和相同的 β 值，则随机变量的组合 $X_1/(X_1+X_2)$ 服从贝塔分布 Beta(α_1, α_2)。

(5) 如果 X 服从总体均值为 μ 和标准差为 σ 的正态分布，则 $Y=\dfrac{(X-\mu)^2}{2\sigma^2}$ 是一个具有参数 α=1/2 的标准伽马变量，即 Y 服从伽马分布 $Gamma(1/2,1)$。

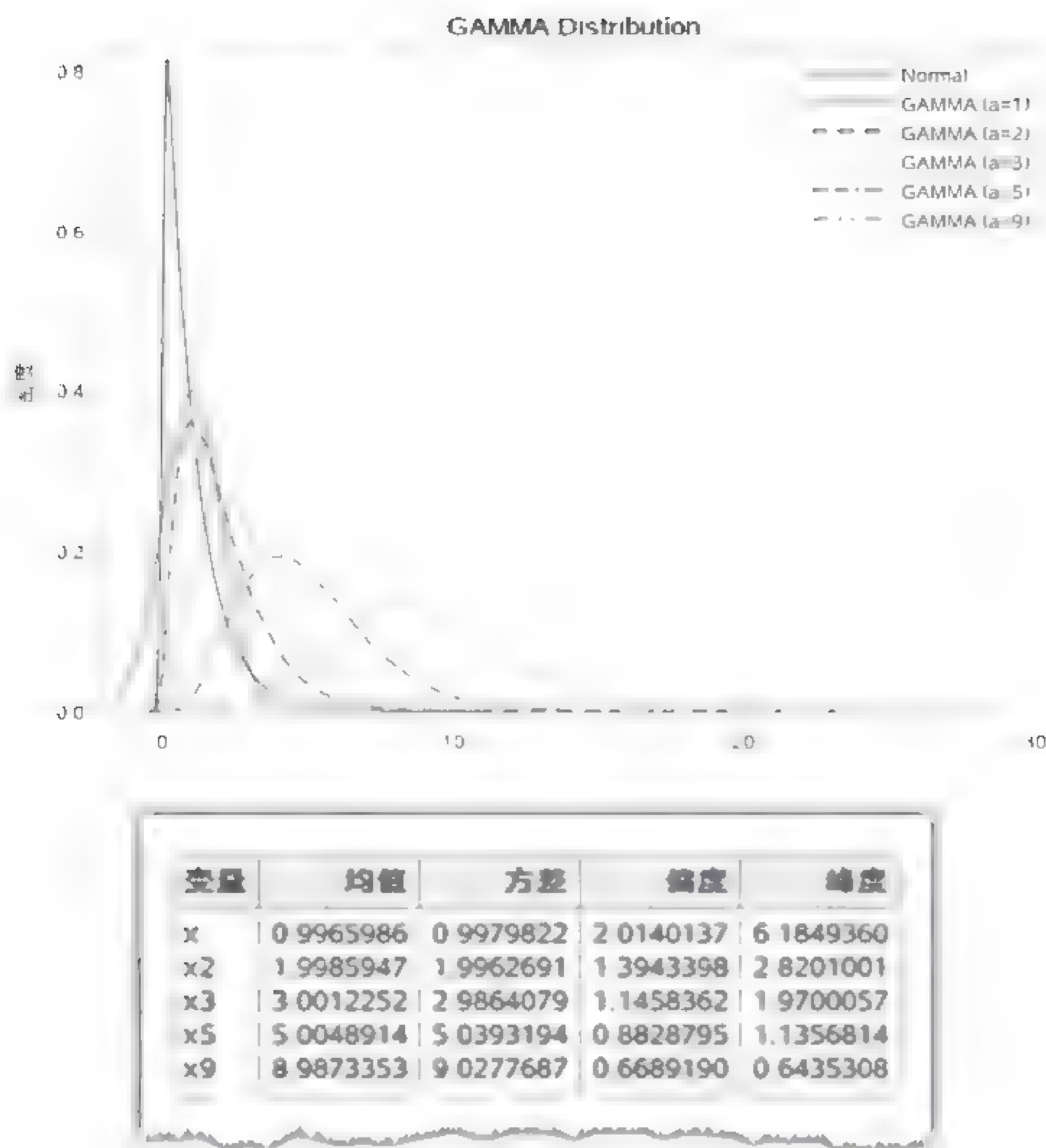
SAS 程序 20-46 用伽马随机数发生器绘制伽马分布曲线如图 20-57 所示。

程序20-46　伽马分布的概率密度曲线和累积分布曲线

```
data sample;
  do N = 1 to 100000;
    x  = rand("GAMMA", 1 );
    x2 = rand("GAMMA", 2 );
    x3 = rand("GAMMA", 3 );
    x5 = rand("GAMMA", 5 );
    x9 = rand("GAMMA", 9 );
    output;
  end;
  keep x x2 x3 x5 x9;
run;

%macro drawline(x, label, pattern);
  density &x / type=kernel legendlabel=&label lineattrs=(pattern=&pattern);
%mend;

proc sgplot data=sample;
  title 'GAMMA Distribution';
  density x / type=normal legendlabel='Normal' lineattrs=(pattern=solid
    color=red);
  %drawline(x,  'GAMMA (a=1)',solid);
  %drawline(x2, 'GAMMA (a=2)',dash);
  %drawline(x3, 'GAMMA (a=3)',dot);
  %drawline(x5, 'GAMMA (a=5)',dashdashdot);
  %drawline(x9, 'GAMMA (a=9)',dashdotdot);
  keylegend / location=inside position=topright across=1;
  xaxis display=(nolabel);
run;
proc means data=sample mean var skewness kurtosis; run;
```



变量	均值	方差	偏度	峰度
x	0.9965986	0.9979822	2.0140137	6.1849360
x2	1.9985947	1.9962691	1.3943398	2.8201001
x3	3.0012252	2.9864079	1.1458362	1.9700057
x5	5.0048914	5.0393194	0.8828795	1.1356814
x9	8.9873353	9.0277687	0.6689190	0.6435308

图 20-57　伽马分布的概率密度曲线

要特别注意图20-57中均值 $\mu=\dfrac{\alpha}{\beta}$ 和方差 $\sigma^2=\dfrac{\alpha}{\beta^2}$，它们在 β=1 时都等于对应的 α 值。

20.3.10　爱尔朗分布

与伽马分布类似，爱尔朗分布（Erlang Distribution）也是从泊松分布导出。对于给定发生率 λ 的泊松分布，在第 n 个泊松事件发生之前的等待时间，其累积分布函数 $D(x)$ 为

$$D(x)=P(X\leqslant x)=1-\frac{\Gamma(h,x\lambda)}{\Gamma(h)}$$

其中 x 为大于等于 0 到正无穷；$\Gamma(x)$ 为完全伽马函数；$\Gamma(\alpha,x)$ 为不完全伽马函数。当 h 为整数时，该分布就是爱尔朗分布。对累积分布函数 $D(x)$ 求微分，即得该分布概率密度函数如下：

$$P(x)=D'(x)=\frac{\lambda(\lambda x)^{h-1}}{(h-1)!}\mathrm{e}^{-\lambda x}$$

爱尔朗分布就是伽马分布在 $\alpha=h$ 为整数时的统计分布，当 $\alpha=1$ 时该统计分布进一步演变为指数分布。服从爱尔朗分布的随机变量可分解为多个同参数的指数分布的随机变

量的和，这一性质使得爱尔朗分布被用于排队模型的分析之中。

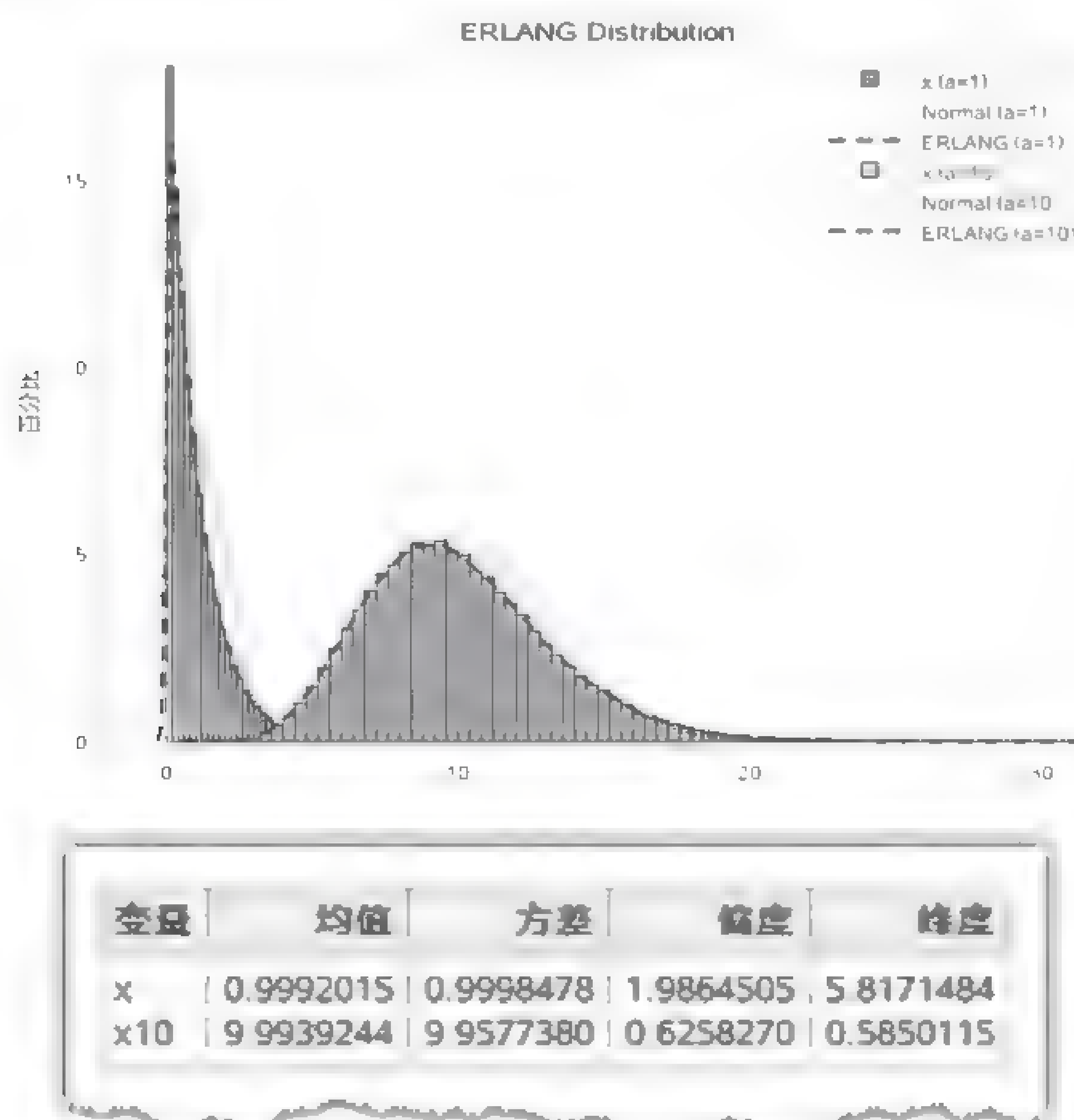
下面为 SAS 绘制的爱尔朗 $a=1$ 和 $a=10$ 时的分布曲线（见程序 20-47 和图 20-58）：

程序20-47 爱尔朗分布a=1和 a=10时的分布曲线

```
data sample;
  do N = 1 to 100000;
    x=RAND('ERLANG', 1);    /*a =1,2,...; x>0*/
    x10=RAND('ERLANG', 10); /*a =1,2,...; x>0*/
    output;
  end;
  keep x x10 ;
run;
proc sgplot data=sample;
  title 'ERLANG Distribution';
  histogram x / legendlabel='x (a=1)' filltype=gradient;
  density x / type=normal legendlabel='Normal (a=1)' lineattrs=(pattern=dot
    color=black);
  density x / type=kernel legendlabel='ERLANG (a=1)' lineattrs=(pattern=dash
    color=blue);

  histogram x10 / legendlabel='x (a=10)' ;
  density x10 / type=normal legendlabel='Normal (a=10)' lineattrs=(pattern=dot
    color=black);
  density x10 / type=kernel legendlabel='ERLANG (a=10)' lineattrs=(pattern=dash
    color=black);
  keylegend / location=inside position=topright across=1;
  xaxis display=(nolabel);

run;
proc means data=sample mean var skewness kurtosis; run;
```



变量	均值	方差	偏度	峰度
x	0.9992015	0.9998478	1.9864505	5.8171484
x10	9.9939244	9.9577380	0.6258270	0.5850115

图 20-58 爱尔朗分布与正态分布

20.3.11　韦布尔分布

韦布尔分布（Weibull Distribution）通常记为 Weibull(α, β)，韦布尔分布的概率密度函数和累积分布函数为

$$P(x)=\alpha\beta^{-\alpha}x^{\alpha-1}e^{-\left(\frac{x}{\beta}\right)^{\alpha}}$$

$$D(x)=1-e^{-\left(\frac{x}{\beta}\right)^{\alpha}}$$

其中 x 为大于等于 0 到无穷。其总体均值、方差、偏度和峰度系数为

$$\mu=\beta\Gamma(1+\alpha^{-1})$$

$$\sigma^2=\beta^2\left[\Gamma(1+2\alpha^{-1})\Gamma^2(1+\alpha^{-1})\right]$$

$$\gamma_1=\frac{2\Gamma^3(1+\alpha^{-1})-3\Gamma(1+\alpha^{-1})\Gamma(1+2\alpha^{-1})+\Gamma(1+3\alpha^{-1})}{\left[\Gamma(1+2\alpha^{-1})-\Gamma^2(1+\alpha^{-1})\right]^{\frac{3}{2}}}$$

$$\gamma_2=\frac{f(\alpha)}{\left[\Gamma(1+2\alpha^{-1})-\Gamma^2(1+\alpha^{-1})\right]^2}$$

其中 $\Gamma(z)$ 为伽马函数。

韦布尔分布是用于求偏离中位数很远的极端或罕见值的极值分布，它最初是用于量化疲劳数据，但也可以用来分析系统中设计最薄弱的环节。韦布尔分布给出了关于物体寿命的分布，被大量用于现实中如风速分布，灾难保险损失，生存分析和精算科学。有时也被称为 Rosin-Rammler 分布，双参数形式的韦布尔分布其位置参数为 0。有些文献中也记为 Weibull(λ, k)，其中 $\lambda=\beta$, $k=\alpha$。程序 20-48 绘制了韦布尔分布的概率密度曲线（见图 20-59），其累计分布曲线如图 20-60 所示。

程序20-48　韦布尔分布的概率密度曲线

```
data sample;
  do x=0 to 2.5 by 0.01 ;
    p=pdf("WEIBULL", x,   1, 0.5);
    p2=pdf("WEIBULL", x,  1, 1);
    p3=pdf("WEIBULL", x,  1, 1.5);
    p4=pdf("WEIBULL", x,  1, 5);
    output;
  end;
run;

proc sgplot;
  title "WEIBULL";
  series x=x y=p / legendlabel="λ=1, k=0.5"
    lineattrs=(pattern=solid color=red);
  series x=x y=p2 / legendlabel="λ=1, k=1 "
    lineattrs=(pattern=dash color=green);
  series x=x y=p3 / legendlabel="λ=1, k=1.5"
    lineattrs=(pattern=dot color=blue); ;
  series x=x y=p4 / legendlabel="λ=1, k=5"
    lineattrs=(pattern=dashdashdot color=red);
  keylegend / location=inside position=topright across=1;
  yaxis max=2;
run;
```

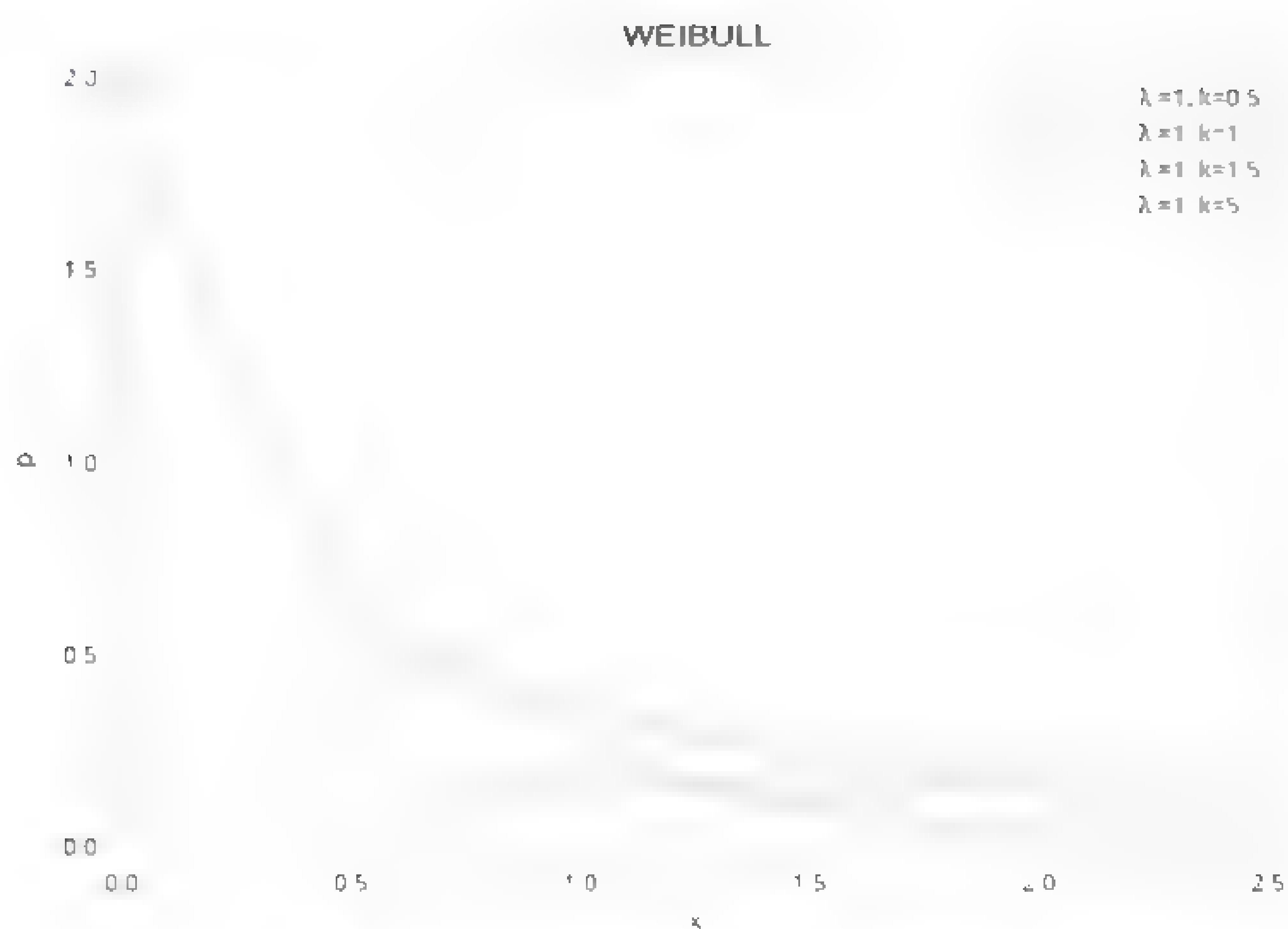


图 20-59　韦布尔分布的概率密度曲线



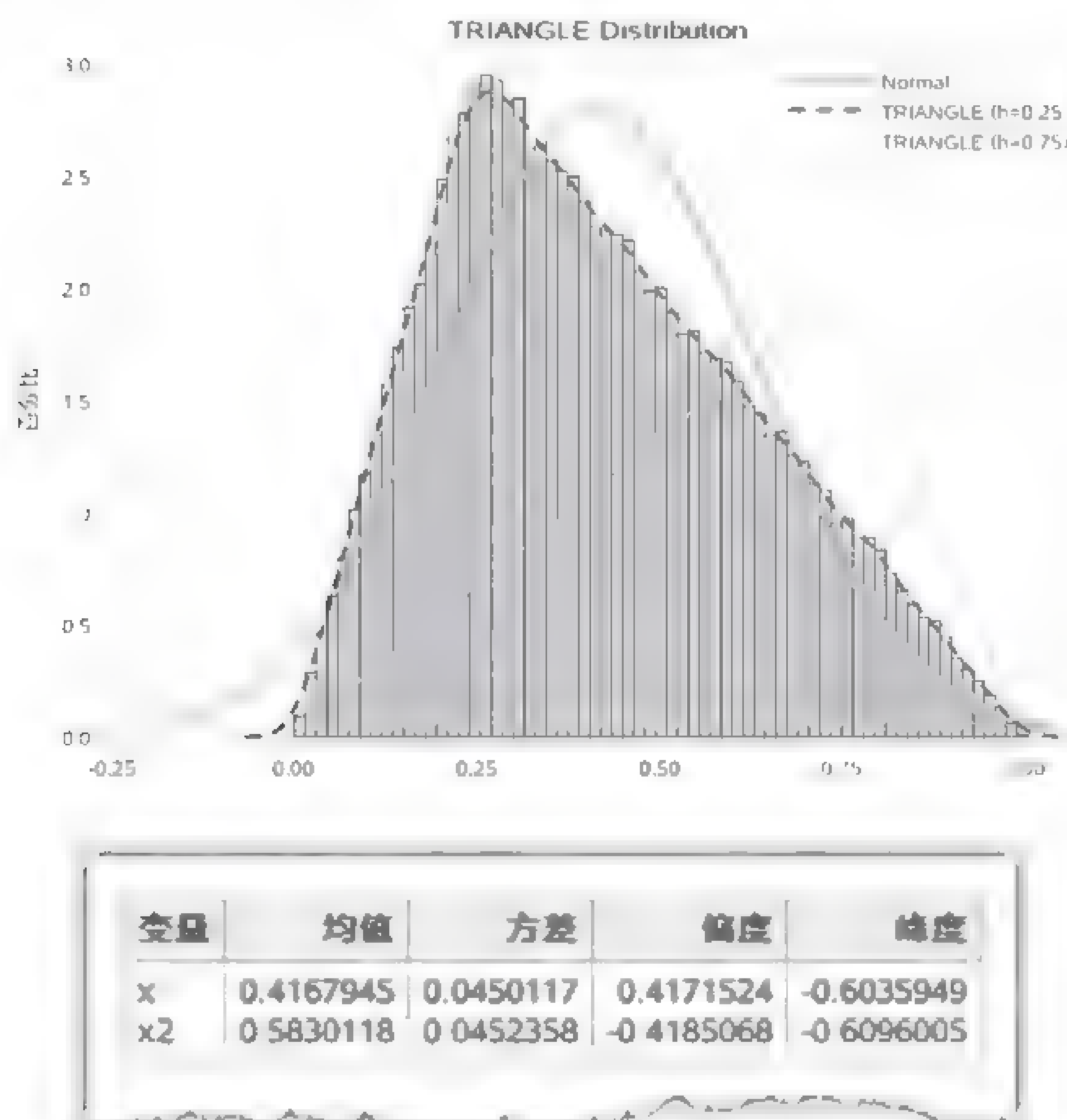
图 20-60　韦布尔分布的累积分布曲线

20.3.12　三角分布

三角分布（Triangular Distribution）就是在 0~1 区间以某个固定概率 $h(0 < h < 1)$ 为顶点构成的三角形（见图 20-61）。三角分布可模拟两个均匀分布的随机变量的均值的分布，也可用于蒙特卡罗仿真对离散系统进行建模。下面的 SAS 程序 20-49 生成服从三角分布的随机变量并绘制了 $h=0.25$ 和 $h=0.75$ 时的三角分布曲线，图中的钟形曲线为参考的正态分布曲线。

程序20-49 三角分布的概率密度曲线

```
data sample;
  do N = 1 to 100000;
    h=0.25;
    x=RAND('TRIANGLE', h) ; /*0 ≤ x ≤ 1; 0 ≤ h ≤ 1 */
    h=0.75;
    x2=RAND('TRIANGLE', h) ; /*0 ≤ x ≤ 1; 0 ≤ h ≤ 1 */
    output;
  end;
  keep x x2 ;
run;
proc sgplot data=sample;
  histogram x / legendlabel='x (a=1)';
  density x   / type=normal legendlabel='Normal'
    lineattrs=(pattern=solid color=red);
  density x   / type=kernel legendlabel='TRIANGLE (h=0.25)'
    lineattrs=(pattern=dash color=blue);
  density x2  / type=kernel legendlabel='TRIANGLE (h=0.75)'
    lineattrs=(pattern=dot);
  keylegend   / location=inside position=topright across=1;
  xaxis display=(nolabel);
  title 'TRIANGLE Distribution';
run;
proc means data=sample mean var skewness kurtosis; run;
```



变量	均值	方差	偏度	峰度
x	0.4167945	0.0450117	0.4171524	-0.6035949
x2	0.5830118	0.0452358	-0.4185068	-0.6096005

图 20-61 三角分布

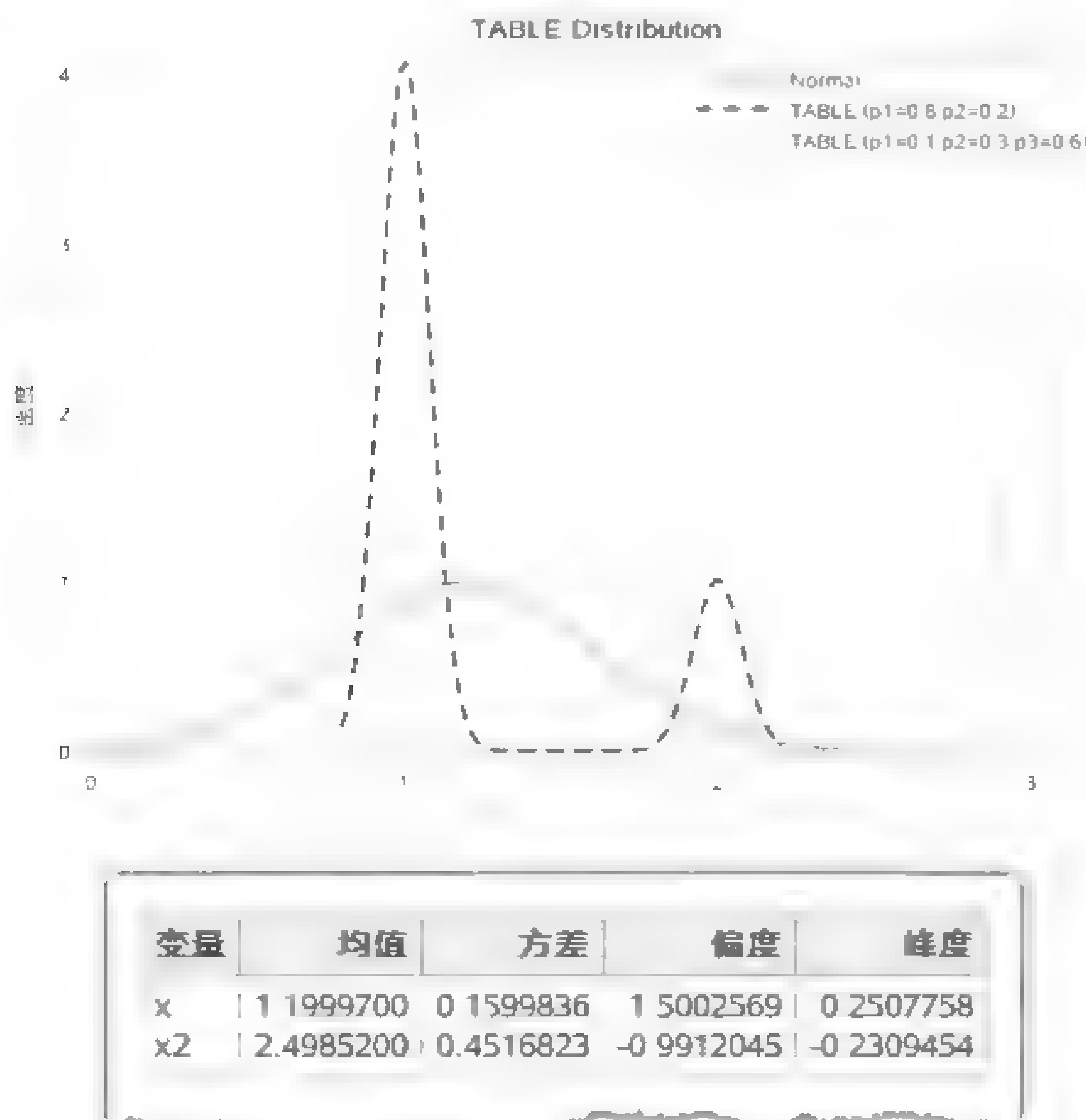
20.3.13 Table 分布

Table 分布（Table Distribution）就是服从给定概率序列的数据分布，它通过指定多

个分段的抽样概率来模拟任何分类数据的分布。在随机数发生器中它可以按照指定的概率生成对应比率的观测。比如给定两个概率值 $p_1=0.8$ 和 $p_2=0.2$，生成包含 10 样本的随机序列，其中数值等于 1 的比例占比为 0.8，而数值等于 2 的比例占比为 0.2（见图 20-62）。Table 分布随机数发生器可在 SAS 中用于构造任何统计分布的数据，从而进行数据模拟和数据分析。程序 20-50 可输出结果如图 20-62 所示。

程序20-50 Table 分布的随机数生成器以及对应的核密度估计曲线

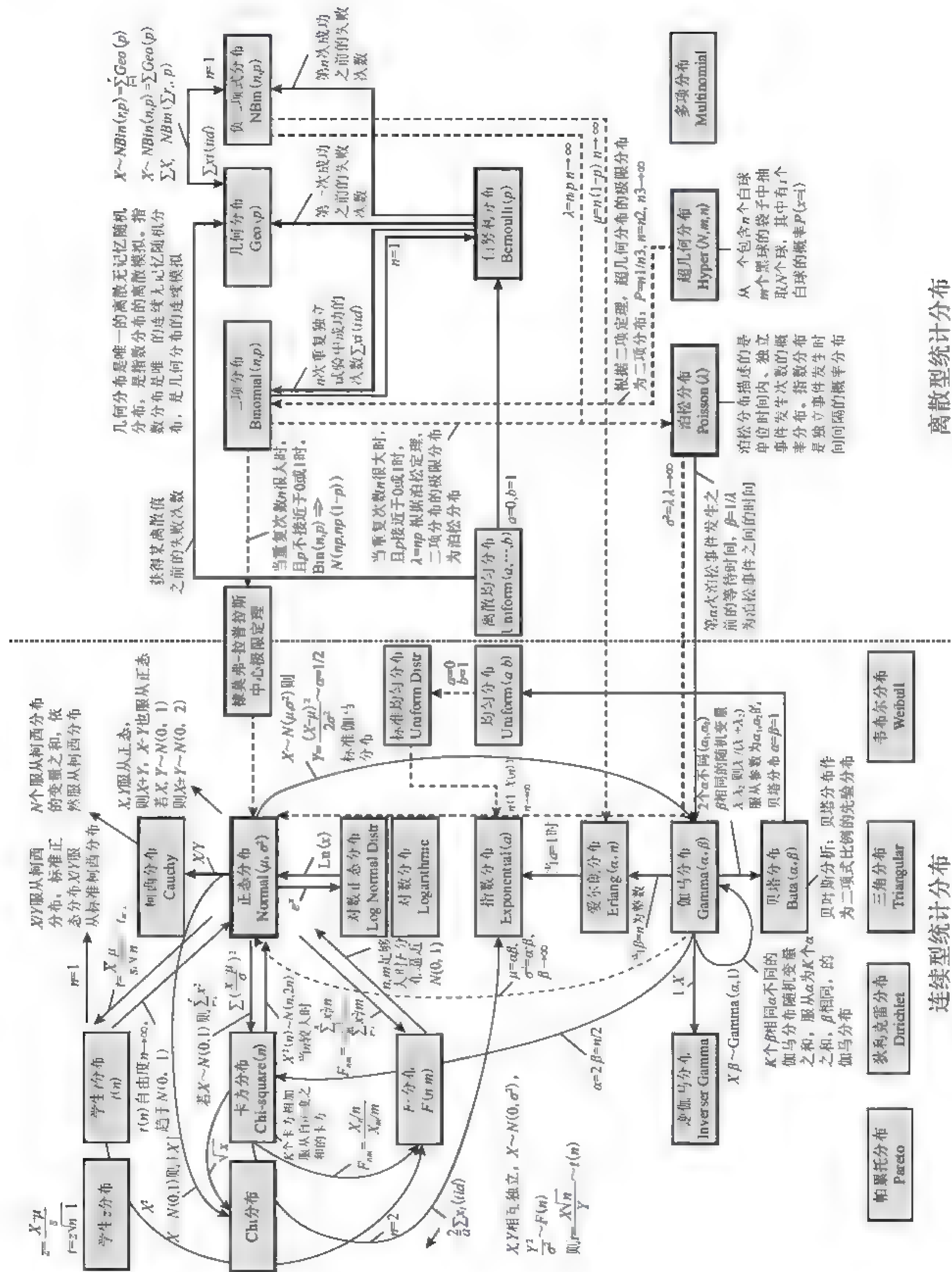
```
data sample;
  do i = 1 to 100000;
    p1=0.8; p2=0.2;
    x=RAND('TABLE', p1, p2 ); /*0 ≤ p1, p2, ... ≤ 1 */
    p1=0.1; p2=0.3; p3=0.6;
    x2=RAND('TABLE', p1, p2, p3); /*0 ≤ p1, p2, ... ≤ 1 */
    output;
  end;
  keep x x2;
run;
proc sgplot data=sample;
  density x   / type=normal legendlabel='Normal'
    lineattrs=(pattern=solid color=red);
  density x   / type=kernel legendlabel='TABLE (p1=0.8 p2=0.2)'
    lineattrs=(pattern=dash color=blue);
  density x2   / type=kernel legendlabel='TABLE (p1=0.1 p2=0.3 p3=0.6)'
    lineattrs=(pattern=dot);
  keylegend   / location=inside position=topright across=1;
  xaxis display=(nolabel);
  title 'TABLE Distribution';
run;
proc means data=sample mean var skewness kurtosis; run;
```



变量	均值	方差	偏度	峰度
x	1.1999700	0.1599836	1.5002569	0.2507758
x2	2.4985200	0.4516823	-0.9912045	-0.2309454

图 20-62　Table 分布

附录：各统计分布之间的关系



第21章

方差分析

利用样本数据推断总体分布特征的数据分析方法称为推断性统计，它与利用样本数据构建统计量的描述性统计共同构成单变量统计分析方法的基础。在推断性统计中，参数估计要解决的问题是利用样本数据如何估计描述总体特征的参数，而假设检验则是利用样本数据来判断对总体的假设是否成立。对于单变量的数据统计分析方法，其内在结构关系如图 21-1 所示。

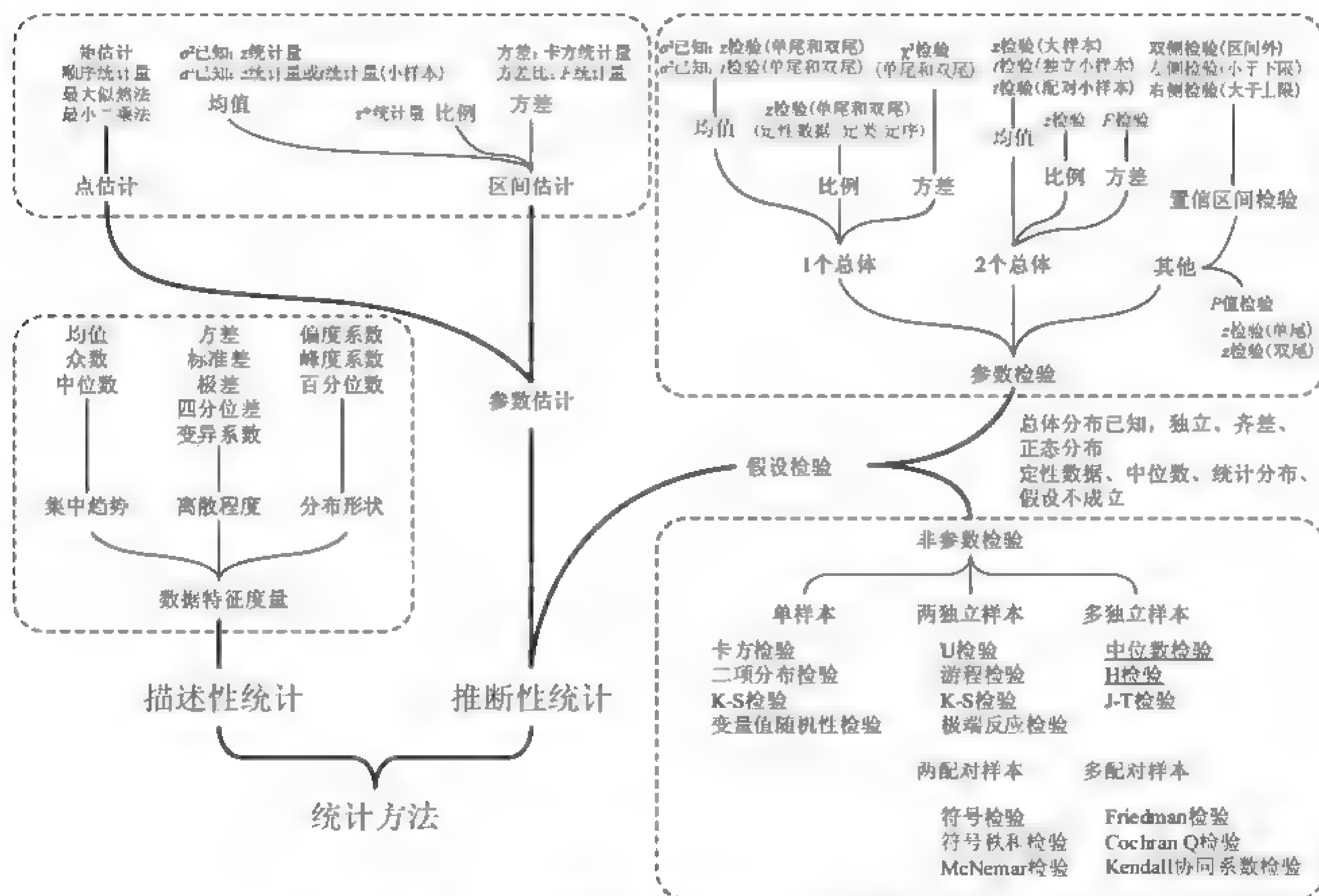


图 21-1　描述性统计与推断性统计结构关系

21.1　假设检验

假设检验的核心思想是利用小概率事件竟然在一次试验中发生来否定对总体参数或分布的假设。因为如果我们对总体参数或分布所作的假设是正确的话，样本值与原假设出现显著差异的可能性应该是很小的。但如果这种差异在一次随机抽样中就显现出来，

我们就有理由怀疑当初的假设是否是真的。也就是说如果假设成立的可能性，小于给定显著性水平 α 所要求的临界值，我们必须拒绝原假设。假设检验包括两部分：

（1）假设是指我们对总体参数或分布特征做出某种假设，在统计学上称为零假设（Null Hypnosis）或原假设，记作 H_0；而与零假设对立的假设称为备择假设，记作 H_1。通常我们收集证据想要否定的假设作为零假设，它是一个包含“等于”（包括大于等于 / 小于等于）含义的命题，比如 μ=3.5 或者 $\mu \geqslant 3.5$ 。而把与零假设陈述的内容相反的命题作为备择假设，它是一个包含“不等于”“小于”或“大于”等含义的命题，比如 $\mu \neq 3.5$ 或者 $\mu < 3.5$。

（2）检验是利用观测到的样本信息来判断上面的假设是否成立，最重要的是对样本信息和原假设的差异进行分析。基于观测到的一个样本，假定已知它服从特定分布，且已知某些总体参数的情况下，我们可以构造出一个特定的检验统计量。由于该统计量基于一次抽样数据计算所得，因此它也是一个随机变量，服从特定分布。

如果样本与假定参数与分布的总体差异小于我们人为指定的水准，称之为“统计不显著”，说明样本和总体之间的差异完全是随机性引起的，我们需要接受零假设，即样本数据服从特定参数的总体分布，否则就说明样本和假定总体之间的差异并非完全是由随机性引起的，差异是显著的，必须拒绝零假设。

我们把人为指定允许出现差异的最低水平，称为显著性水平，记作 α；而 $1-\alpha$ 则称为置信水平。在图 21-2 所示的抽样分布图中，它们在横坐标上所对应的样本统计量区间，分别称之为拒绝域和接收域。由于显著性水平 α 是一个人为指定的常量，只要统计量落在拒绝域我们就拒绝原假设，它根本无法给出观测数据（样本）与原假设之间差异的程度，因此统计学家 R. A. Fisher 引入“P- 值”来衡量当原假设为真时，得到与样本一样的观测值或者更加极端值的概率。

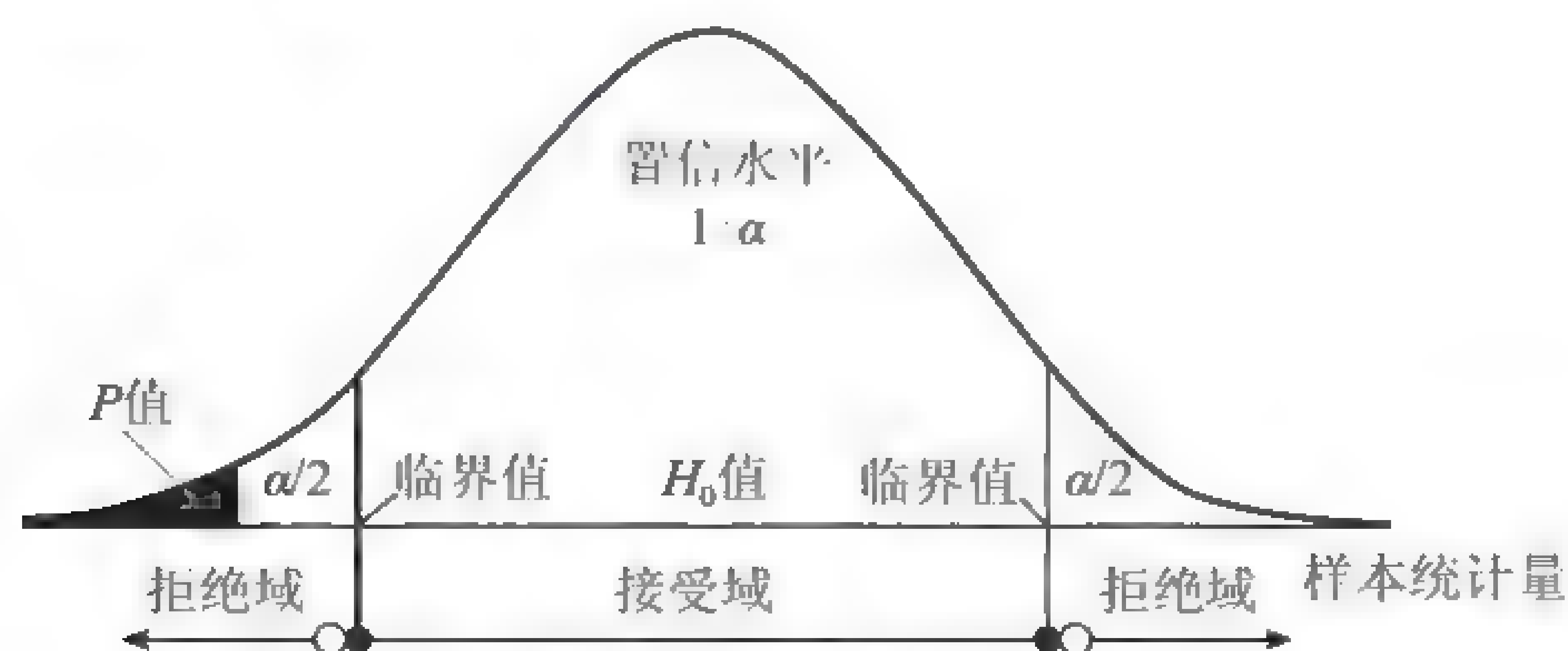


图 21-2 假设检验的原理

P- 值的本质就是观测到的显著性水平，也是零假设 H_0 能被拒绝的最小 α 值。P- 值越小，结果差异的显著性就越大，拒绝 H_0 的理由就越充分。注意，显著性水平 α 是理论上要求的显著性水平，而 P- 值是基于样本计算出来的值，因此 P- 值就是用实际样本拒绝原假设的那个最小的 α 值，在许多研究领域 P- 值 < 0.05 通常被认为是可接受的临界水平，与人为指定的 α 值的临界水平 0.05 类似。

在假设检验中有两种统计学错误需要加以区分。

（1）如果零假设事实上成立，但统计检验的结果不支持该零假设（否定零假设），则此类错误称为Ⅰ型错误，也称假阳性错误，它在科学研究中容易导致错误的科学论断；Ⅰ型错误的发生概率记为α，就是显著性水平。

（2）若零假设事实上不成立，但统计检验结果支持零假设（接受零假设），此类错误称为Ⅱ型错误，也称假阴性错误，它在科学研究中可能导致错失科学发现。Ⅱ型错误发生的概率记为β。

如果错误的假设导致错误的推断，即假设本身都是错的，就称为Ⅲ型错误，Ⅲ型错误并非统计学错误。

比如一个人用验孕棒检验是否怀孕，如果此人没有怀孕但验孕棒指示她怀孕了，则这是Ⅰ型错误；如果此人确实怀孕了但验孕棒指示并未怀孕，则这是Ⅱ型错误。Ⅱ型错误与取样范围、试验设定标准的高低和效果有关。假设此人怀孕了，然后你推断此人没来上班，但实际上此人正在上班，则此类错误称为 Ⅲ 型错误。统计决策中不可能同时减少 Ⅰ 型错误和Ⅱ 型错误，也就是说α和β不可能同时减少而是此消彼长，它们在统计检验过程中的关系如表 21-1 所列。

表 21-1　统计检验过程

H_0 检验		实际情况	
		H_0 为真	H_0 为假
根据研究结果判断	接受 H_0	正确判断（1—α）	Ⅱ型错误（β）
	拒绝 H_0	Ⅰ型错误（α）	正确判断（1—β）

21.2　方差分析

方差分析是一种特殊的假设检验，它用来判断多组数据的总体均值之间的差异是否显著。对两组数据比较一般使用学生 *t*- 检验即可，但 *t*- 检验在多重比较时引发Ⅰ型错误的机会增高，所以在比较多组样本数据所代表的总体之平均值是否有差异时，往往采用方差分析方法（Analysis of Variance ，ANOVA）。方差分析是主要探讨连续型变量，在类别型自变量不同水平（也称为因素的具体表现）下的观测值，来检验各类别所属总体的均值是否相等的统计模型，广泛用于实验数据分析中。

21.2.1　学生*t*-检验

1. 统计量计算

对某个衡量差异的变量计算 *t* 统计量有多种方法，最简单的是利用 PROC MEANS 直接计算 *t* 统计量及其 *P*- 值 (Pr > |*t*|)。程序 21-1 的例子中假定变量 *d* 是关于某变量的两个样本的差异值，我们定义原假设 H_0 为这两个总体的均值相等，即 μ_1 μ_2。而图 21-3 为该 SAS 程序对此差异值 *d* 计算出来的 *t* 统计量为 0，*P*- 值等于 1.00，它说明我们需要

接纳原假设 H_0，即 x，y 这两个变量描述的两个总体均值是相等的（其实 x 和 y 两个样本只是顺序不同）。但如果计算出来的 P- 值小于 0.05 则需要拒绝零假设。

程序21-1 两个样本差异值的简单t- 检验

```
data mydata;
  input x y @@;
  d=x-y;
  datalines;
1 5
2 4
3 3
4 2
5 1
run;
proc means data=mydata t probt;
  var d;
run;
```

分析变量：d

t 值	Pr > |t|
0.00	1.0000

图 21-3 学生 t 统计量

程序 21-2 代码也能输出类似的结果，不过包含更多的信息，包括分布图和Q－Q图等。

程序21-2 两个样本差异值的t-检验

```
proc ttest data=mydata;
  var d;
run;
```

2. t-检验类型

t- 检验方法主要用于样本容量小于 30 且总体标准差 σ 未知时判断一个样本的均值和总体均值的差别（或者两个样本均值的差别）是否具有统计学意义，即差别显著。包括单个总体的 t- 检验，两个总体的 t- 检验（独立样本 t- 检验和配对样本 t- 检验）。

有多种方法可以实现单个总体的 t- 检验，程序 21-3 的代码是基于原始数据通过指定 mu0 参数进行的 t- 检验，指定不同的 mu0 值会得到不同的检验结果。而图 21-4 所示的结果中，由于 P- 值 0.9754 不小于 0.05，我们不能拒绝 sashelp.class 身高的样本均值和所指定的总体均值 62.3 相等的原假设，也就是说我们应该接受样本均值跟总体均值 62.3 之间在统计学上没有显著差异的假设。

程序21-3 样本跟指定总体mu0 的t-检验 (UNIVARIATE)

```
proc univariate data=sashelp.class normal mu0=62.3;
  var height;
run;
```

位置检验：Mu0=62.3

检验	统计量		p 值	
Student t	t	0.031322	Pr > |t|	0.9754

图 21-4 样本与指定总体均值的学生 t- 检验

程序 21-4 的 SAS 代码也能输出等价的结果，但此时是用 H_0 选项进行指定的。

程序21-4　样本跟指定总体 H0 的*t*-检验 (TTEST)

```
proc ttest data=sashelp.class H0=62.3;
  var height;
run;
```

对于来自不同总体的两个样本的 *t*- 检验，TTEST 过程步只需要有个用 CLASS 语句指定的分类变量标记分组（比如用 Sex 变量标记两个不同性别组），即可用程序 21-5 代码进行双样本 *t*- 检验：

程序21-5　双样本*t*-检验

```
proc ttest data=sashelp.class alpha=0.05;
  class sex;
  var height;
run;
```

而对于配对样本的 *t*- 检验，比如对医学研究中受试验对象处理前后的两组数据进行 *t*- 检验，可以有两种方法进行。比如我们将 SASHELP.CLASS 中的两个变量 Height 和 Weight 分别对应两个配对样本，现在进行配对样本 *t*- 检验，可用程序 21-6 代码进行：

程序21-6　配对样本*t*-检验

```
proc ttest data=sashelp.class;
  paired weight * height;
run;
```

我们也可以先计算两个变量的差异值 *d*，然后对衡量差异的变量 *d* 做单变量分析，用检查输出的 *t* 统计量及其 *P*- 值来做配对样本 *t*- 检验。

程序21-7　计算差异值然后利用单变量分析

```
data mydata;
  set sashelp.class;
  d=weight-height;
run;
proc univariate data=mydata normal ;
  var d;
run;
```

在 SAS 中 MEANS、UNIVARIATE 和 TTEST 3 个过程步都可以进行 t 检验，读者可以查看帮助文档了解更多细节。本书附录 4 列出了 t 分布临界值表可供检验之用。

21.2.2　单因子方差分析

方差分析的主要工具是由 R.A Fisher 发明的 F 统计量及 F 分布，它利用误差平方和与自由度相除来计算均方差以衡量样本的数据变异，然后用组间均方差和组内均方差的比值构造出 F 统计量。通过比较 F 统计量与对应显著性水平下的 F 临界值来判断各因素水平下的表现是否存在显著差异。

方差分析根据因子数量的多少可以分为单因子方差分析、双因子方差分析以及多因子方差分析。根据因子的特性不同，也可以分为固定效应的方差分析、随机效应的方差

分析以及混合效应的方差分析；随机效应的方差分析所考虑的因子是来自于所有可能总体的一组样本，但该因子方差分析所推论的并非是所选定的因子，而是因子背后的总体。

进行方差分析必须符合3个前提条件：各组样本必须是独立的；各组样本所代表的总体必须服从正态分布或者近似正态分布；各组的方差还必须相等，满足方差齐性。在具体操作上，方差分析的基本步骤为：

（1）假定有 k 组数据，每组数据的样本大小为 n_i(其中 $i=1, 2, \cdots, k$)，各组数据的样本量可以相等，也可能因实际条件限制并不相等。

（2）计算总的离差平方和（Sum of Squares for Total，SST）：它等于所有观测值 x_{ij} 跟全部数据均值 $\bar{\bar{x}}$ 之差的平方和，反映的是全部数据的离散情况，也就是全部样本数据对总体均值的偏离程度。SST也有文献写作TSS -Total Sum of Squares。其计算公式为

$$SST=\sum_{i=1}^{k}\sum_{j=1}^{n_i}(x_{ij}-\bar{\bar{x}})^2$$

（3）计算误差项的平方和（Sum of Squares for Error， SSE）：它等于各组样本的观测值 x_{ij} 跟本组样本均值 $\bar{x}_i$ 之差的平方和。它反映的是各组数据内部观测值与该组数据均值之间的离散程度，也称为组内离差平方和（Within Sum of Squares，WSS）。组内离差平方和反映从来自相同总体的不同样本在抽样时引入的随机误差大小。其计算公式为：

$$SSE=\sum_{i=1}^{k}\sum_{j=1}^{n_i}(x_{ij}-\bar{x}_i)^2$$

（4）计算水平项平方和（Sum of Squares for Factor A，SSA）：它等于各组样本均值 $\bar{x}_i$ $(i=1, 2, \cdots, k)$ 跟总体均值 $\bar{\bar{x}}$ 的离差平方和。它反映来自同一总体的各组样本的均值之间的差异程度，也称为组间平方和（Between Sum of Squares，BSS）。它的构成既包括因抽样造成的随机误差，也包括由于因子本身所导致的系统误差。其计算公式为

$$SSA=\sum_{i=1}^{k}\sum_{j=1}^{n_i}(\bar{x}_i-\bar{\bar{x}})^2=\sum_{i=1}^{k}n_i(\bar{x}_i-\bar{\bar{x}})^2$$

上面3种离差平方和之间存在如下关系：

总的离差平方和SST= 误差项离差平方和SSE+ 水平项离差平方和SSA

（5）定义零假设 H_0：假设各组数据来自同一总体，则它们的均值相同，即 $\mu_1=\mu_2=\cdots=\mu_k$ 成立，表示没有系统误差。如果系统误差趋于0，则由随机误差和系统误差构成的组间离差平方和SSA应该会跟反映随机误差大小的SSE相近。

（6）由于3个离差平方和自由度不同，我们需要消除自由度的影响。其中SST的自由度为所有组样本数总和 $n-1$，组间平方和SSA的自由度为因子水平 $k-1$，而组内平方和SSE的自由度为 $n-k$，3个自由度之间也存在关系 $df_{sst}=df_{sse}+df_{ssa}$。因此对于前面计算出来的3种离差平方和，分别对各自的自由度进行分摊，可以消除观测值的多少对离差平方和的影响，从而建立反映组间变异和组内变异的为均方差MSA和MSE，也有文献写作BMSS和WMSS，即Between Means Sum of Squares 和 Within Means Sum of Squares。

$$MSA=\frac{SSA}{df_{ssa}} \text{ 和 } MSE=\frac{SSE}{df_{sse}}$$

我们定义统计量 F 为组间变异和组内变异的比值：

$$F=\frac{MSA}{MSE}=\frac{SSA/df_{ssa}}{SSE/df_{sse}}$$

则当零假设 H_0 为真时，统计量 F 服从分子自由度为 $k-1$，分母自由度为 $n-k$ 的 $F(k-1, n-k)$ 分布。表 21-2 列出了单因素方差分析计算的基本结构。

表 21-2 单因素方差分析的计算

方差来源	平方和 SS	自由度 DF	均方差 MS	统计量 F 值
组间（因子）	SSA	$df_{ssa}=k-1$	MSA	$F=\frac{MSA}{MSE}$
组内（误差）	SSE	$df_{sse}=n-k$	MSE	
总和	SST	$n-1$		

（7）统计决策：将计算出来的统计量 F 与给定的显著性水平 α 下查表（参见本书附录 6：F 分布临界值表）所得的临界值 F_α 进行比较，从而做出接受或拒绝零假设 H_0 的统计决策。通常情况下：

①如果 $F<F_{0.05}$（或对应 P- 值 >0.05），则接纳零假设，即各组样本之间的差异不显著。

②如果 $F_{0.05}\leqslant F<F_{0.01}$（或对应 $0.01<P$- 值 $\leqslant 0.05$），则拒绝零假设，认为各组样本之间的差异是显著的，标记为 *。

③如果 $F\geqslant F_{0.01}$（或对应 P- 值 $\leqslant 0.01$），则拒绝零假设，认为各组样本之间的差异极其显著，标记为 **。

F 分布接纳零假设和拒绝零假设的域如图 21-5 所示。

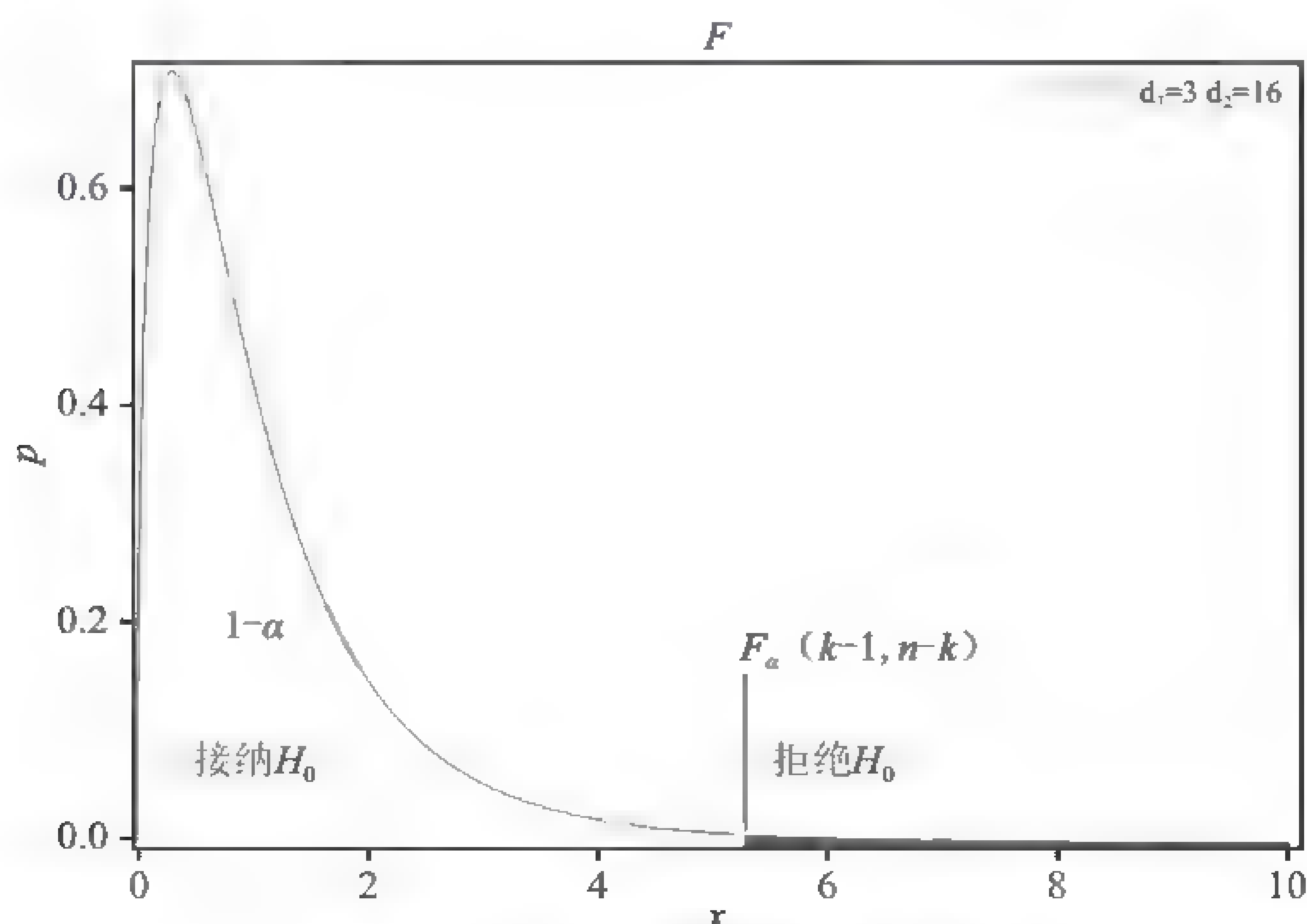


图 21-5 F 分布的接纳域和拒绝域

- 实例分析1：比如某公司研发了4种新型饮料，假定除颜色外其他可能影响销售量的因素（如营养含量、味道、价格）全部相同，现在从某地区经营规模相似的5家超市收集了上个月的销售量如表21-3所列，现在要求分析饮料的颜色对销售量是否产生显著影响？

表 21-3 4 色饮料在 5 家超市的销售数据

超 市	A_1 无色	A_2 粉色	A_3 橘黄色	A_4 绿色
1	26.5	31.2	27.9	30.8
2	28.7	28.3	25.1	29 6
3	25.1	30.8	28.5	32 4
4	29.1	27.9	24.2	31.7
5	27.2	29 6	26.5	32.8

虽然我们可以用SAS的PROC过程步直接进行方差分析，但为了演示计算过程，我们可在DATA步中自己编程计算F统计量并进行检验，完整流程如程序21-8所示。

程序21-8 自己编程计算F统计量并进行检验

```
data mydata;
  input c1-c4;
  datalines;
26.5  31.2  27.9  30.8
28.7  28.3  25.1  29.6
25.1  30.8  28.5  32.4
29.1  27.9  24.2  31.7
27.2  29.6  26.5  32.8
;
data _null_;
  array d[5,4] _temporary_;

  /*1)读入数据到数组中进行打印和后续运算*/
  array r[4] c1-c4;
  do j=1 to n;
    set mydata point=j nobs=n;
    do i=1 to dim(d,2);
      d[j,i]=r[i];
      put d[j,i] @@;
    end;
    put;
  end;
  put "Step1: 读入数据完毕";

  /*2)计算总体均值 m_all 和组内均值 m[i]*/
  m_all=0;
  array m[4] _temporary_;
  do i=1 to dim(d,2);
    m[i]=0;
    do j=1 to dim(d,1);
      m[i]=m[i]+ d[j,i];
    end;
    m all=m all+m[i];
    m[i]=m[i]/dim(d,1);
  end;
  m all m all/ (dim(d,1) * dim(d,2));
  put "Step2: 计算总体和各组均值完毕";
```

```
  /*3)计算 SST, SSE 以及 SSA */
  sst=0;
  sse=0;
  do i=1 to dim(d,2);
    do j=1 to dim(d,1);
    sst=sst+ (d[j,i]-m_all)**2;
    sse=sse+ (d[j,i]-m[i])**2;
    end;
  end;
  ssa=sst - sse;
  put "Step3: 计算总体、组内、组间力差平方和. " SST= SSE= SSA=;

  /*4) 计算自由度，均方差以及 F 统计量*/
  k=dim(d,2); n=dim(d,1)*dim(d,2);
  df1=k-1; df2=n-k;
  msa=(ssa/df1);
  mse=(sse/ df2);
  F=(ssa/df1) / (sse/ df2);
  put "Step4: 计算 F 统计量. " F=;

  /*5) 基于 F 统计量进行统计决策*/
  F005=FINV( 1- 0.05,  df1, df2); /*查 F 临界值表*/
  F001=FINV( 1- 0.01,  df1, df2);
  put "Step5: 给定显著性水平 0.05 和 0.01 的临界 F 值: " F005= F001=;
  put;
  if F<F005               then put "接纳零假设:组间差异不显著";
  if F005<=F AND F<F001   then put "拒绝零假设:组间差异显著 *";
  if F001<=F              then put "拒绝零假设:组间差异极其显著 **";
  stop;
run;
```

系统输出为

```
Step1: 读入数据完毕
Step2: 计算总体和各组均值完毕
Step3: 计算总体、组内、组间力差平方和. sst=115.9295 sse=39.084 ssa=76.8455
Step4: 计算 F 统计量. F=10.486200662
Step5: 给定显著性水平 0.05 和 0.01 的临界 F 值: F005=3.2388715175 F001=5.2922140455
拒绝零假设: 组间差异极其显著 **
```

SAS 程序 21-8 展示了单因素方差分析的内部计算细节，熟悉计算逻辑后用户可用 SAS 封装好的一系列的 PROC 步来完成数据分析过程。用 SAS PROC 步来进行方差分析，首先需要构建 SAS PROC 支持的数据组织形态，步骤如下：

首先对前面的数据进行变换，将多列数据变成两列的分组数据（见程序 21-9），其中 factor 列为因子水平，取值为 A_1、A_2、A_3、A_4；x 列为观测值（见图 21-6）。

程序21-9　将多列数据转化为分组数据

```
data mydataG;
  set mydata;
  array c[4] c1-c4;
  do j=1 to 4;
    factor="A" || trim(left(j));
    x=c[j];
    output;
  end;
  keep factor x;
run;
```

Obs	c1	c2	c3	c4
1	26.5	31.2	27.9	30.8
2	28.7	28.3	25.1	29.6
3	25.1	30.8	28.5	32.4
4	29.1	27.9	24.2	31.7
5	27.2	29.6	26.5	32.8

Obs	factor	x
1	A1	26.5
2	A2	31.2
3	A3	27.9
4	A4	30.8
5	A1	28.7
17	A1	27.2
18	A2	29.6
19	A3	26.5
20	A4	32.8

图 21-6 重组数据供分析

如前所述，方差分析需要符合 3 个前提条件：数据相互独立、正态分布或近似正态分布、总体方差相等，因此我们需要检查数据分布的正态性和方差齐性。首先检查数据分布的正态性（见程序 21-10）。

程序21-10 检测数据分布是否符合正态分布

```
proc univariate data=mydataG normal;
  title "单因素方差分析";
  var x;
  class factor;
run;
```

SAS 自动为各组数据计算了各种有关统计量和描述性分析结果，在每组数据的“正态性检验”下，我们可以看到 *Shapiro-Wilk* 检验的 W 统计量和 p- 值，其中如果 p- 值（Pr < W）大于 0.05 则表明数据符合正态分布。而 4 组数据的 *P*- 值分别为：0.7503, 0.4825, 0.7128, 0.7532，说明各组数据都符合正态分布（见图 21-7）。

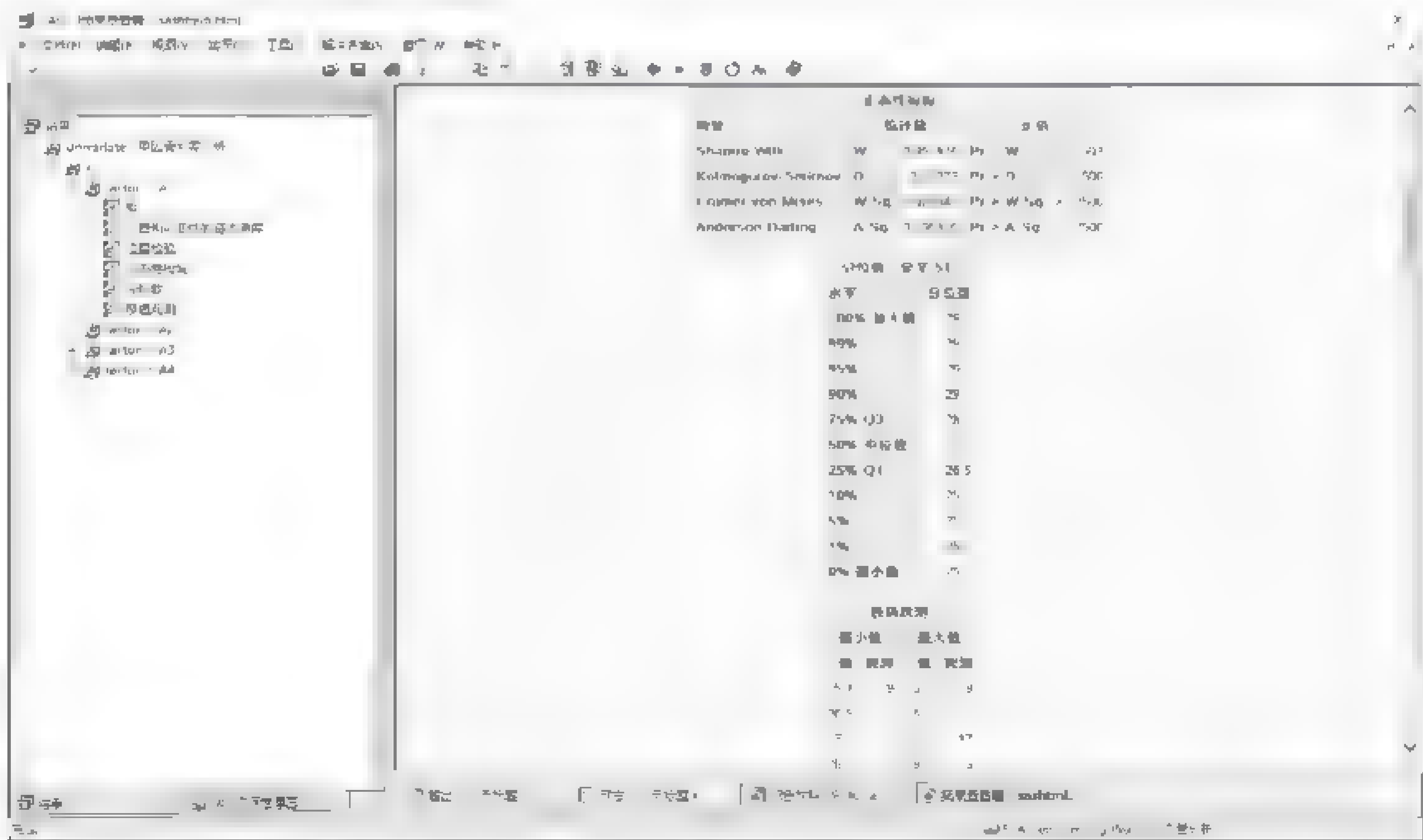


图 21-7 正态性检验

使用 SAS 进行单因子方差分析，可以直接用如下过程步计算（见程序 21-11），它与前面自己用 DATA 步计算得到的结果相同（见图 21-8）。

程序21-11 ANOVA 单因子方差分析
```
proc anova data mydataG ;
  class factor;
  model X = Factor ;
run;
```

源	自由度	平方和	均方	F 值	Pr > F
模型	3	76.8455000	25.6151667	10.49	0.0005
误差	16	39.0840000	2.4427500		
校正合计	19	115.9295000			

R 方	变异系数	均方根误差	x 均值
0.662864	5.446698	1.562930	28.69500

源	自由度	Anova 平方和	均方	F 值	Pr > F
factor	3	76.84550000	25.61516667	10.49	0.0005

图 21-8 单因子方差分析

下面我们也可使用 GLM 过程步来执行方差分析，检验 4 个颜色组的销售量是否存在差异（见程序 21-12 和图 21-9）。

程序21-12 GLM 单因子方差分析
```
proc glm data=mydataG;
  class factor;
  model x=factor ;
  means factor /hovtest snk;
run;
quit;
```

在“方差分析”下的“总体 ANOVA”中，我们可以看到完整的单因素方差分析的结构，其中模型 F 值 10.49 就是我们前面自己计算的 F 统计量。然后我们检查 Pr > F 的值为 0.0005，远小于 p- 值 0.01，因此我们必须拒绝零假设，认为 4 个颜色组的销售量存在显著差异，它们来自不同的总体。

GLM 过程

因变量: x

源	自由度	平方和	均方	F 值	Pr > F
模型	3	76.8455000	25.6151667	10.49	0.0005
误差	16	39.0840000	2.4427500		
校正合计	19	115.9295000			

图 21-9 GLM 过程步输出

其中 means 语句的 hovtest 选项用来做 Levene 方差齐性检验，判定标准是 P- 值（Pr > F 值），如果它大于 0.05 则说明满足方差齐性要求。图 21-10 表示该数据通过 Levene 方差齐性检验 Pr > F 值为 0.6387, 大于 0.05 说明数据符合方差分析的前提条件。另外，GLM 过程步还会画出 x 变量在每个因子上的分布情况，盒形图可揭示各组内部数据的分布特征（见图 21-10）。

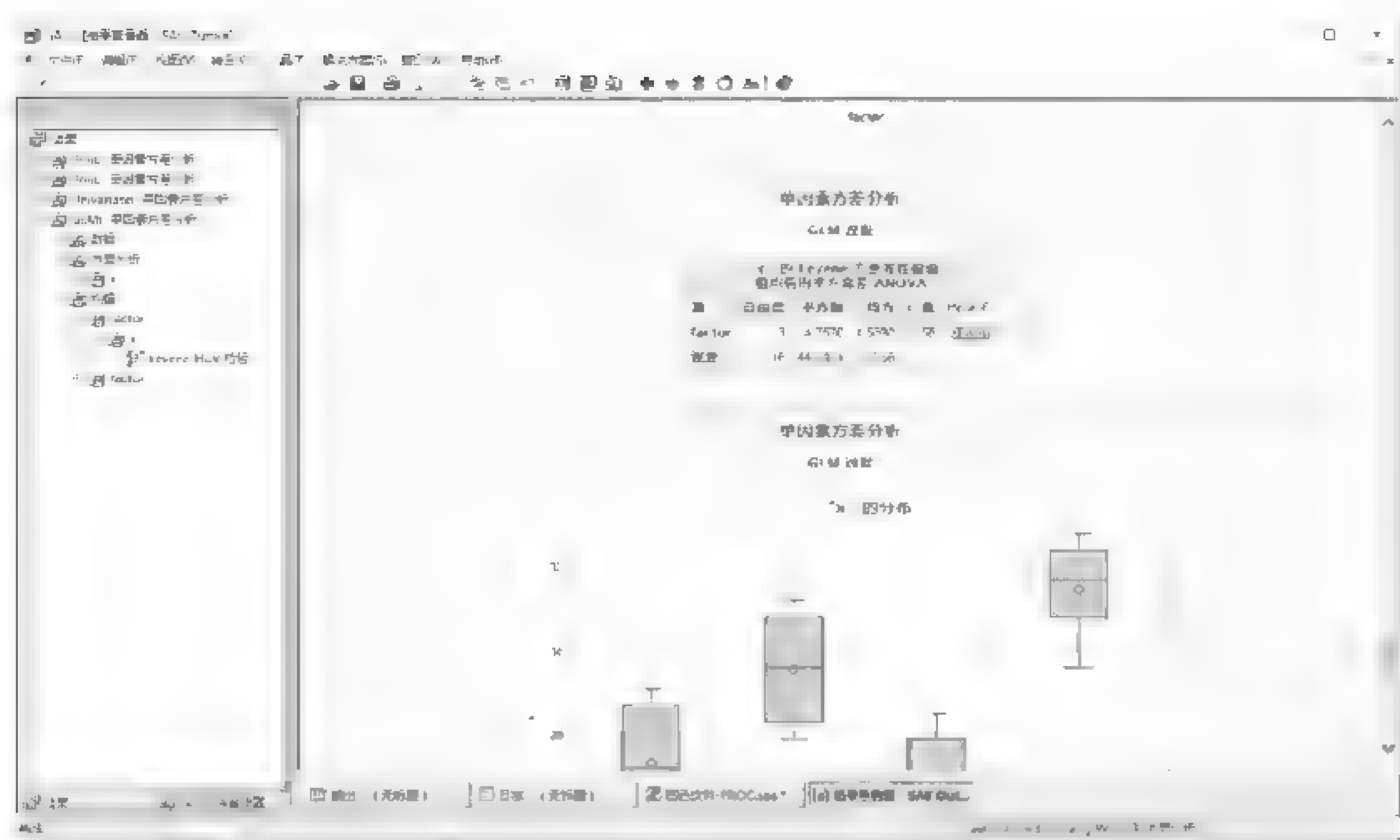


图 21-10 Levene 方差齐性检验

虽然我们知道各组的均值不同，但要知道细节必须进行多重比较。我们可以使用 LSD 方法或者 LSR 方法之一进行检验。其中 Student-Newman-Keuls (SNK) 方法可对 x 变量进行 Newman-Keuls 检验，它属于最小显著极差法 (Least Significant Ranges,LSR) 的改进类型。该方法在完全零假设（不是部分零假设）下可控制 I 型试验错误率（Experiment-Wise Error Rate, EER）。

从下面的方差分析多重比较结果（见图 21-11）看出，因子 A_2, A_4 来自同一个 SNK 分组 A，而 A_1, A_3 则来自另一个 SNK 分组 B。这就是说具有相同 SNK 分组字母的各组数据之间是不存在显著性差异，而具有不同字母的数据则存在显著性差异。它说明 A_1, A_3 可能来自一个总体，而 A_2, A_4 来自另一个总体，它们的总体均值的差异表明，A_2, A_4 (分别表示粉色和绿色) 饮料的销售显然比 A_1, A_3 (无色、橘黄色) 要好。

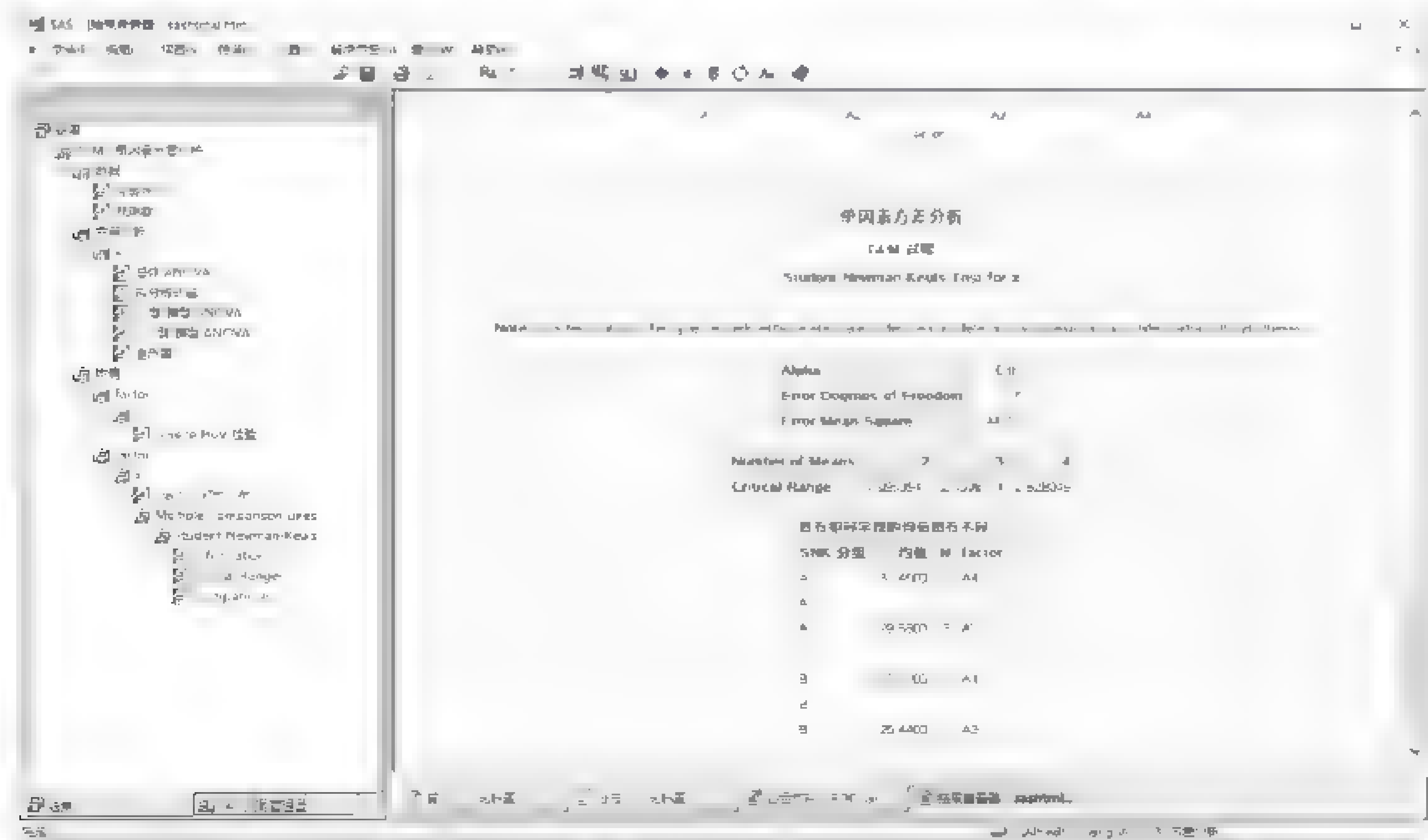


图 21-11 多重比较分析结果

- 实例分析 2:当各组数据的观测数不一样时，我们也可以使用同样的方法进行方差分析。下面的数据为消费者协会收集到的关于零售业、旅游业、民航业和家电制造业的消费者投诉次数。由于行业差异，收集到的数据量是不同的。与前面的方法一样，我们先变换数据然后调用SAS PROC进行方差分析（见程序 21-13 和输出图21-12）。

程序21-13　消费者对零售、旅游、民航和家电制造等4行业的投诉次数分析

```
data mydata;
    input c1-c4;
    datalines;
57  62  51  70
55  49  49  68
46  60  48  63
45  54  55  69
54  56  47  60
53  55   .   .
47   .   .   .
;
proc print data=mydata; run;
```

Obs	c1	c2	c3	c4
1	57	62	51	70
2	55	49	49	68
3	46	60	48	63
4	45	54	55	69
5	54	56	47	60
6	53	55		
7	47			

图 21-12　非平衡数据

利用程序 21-12 的 GLM 代码对该数据进行方差分析，结果输出如图 21-13 所示。

GLM 过程

因变量: x

源	自由度	平方和	均方	F 值	Pr > F
模型	3	845.217391	281.739130	14.79	<.0001
误差	19	362.000000	19.052632		
校正合计	22	1207.217391			

图 21-13　非平衡数据的方差分析

计算结果的 F 值为 14.79，查 F_α 临界值表 $F_{0.01}(3, 19)$ 5.0。由于 F 值大于 $F_{0.01}$ 我们拒绝零假设，也就是说 4 个行业的服务质量具有显著差异。另外，我们也可以直接从图 21-13 中检查 P- 值（Pr > F 值）<.0001，该值小于临界 P- 值 0.01，我们拒绝零假设，4 组样本并非来自同一总体，它们具有显著差异。

- *基于t-检验的 LSD 方法*

为了更进一步了解各组差异的细节，除了我们前面提到的 SNK 多重比较方法，也可以用经典的 Fisher 最小显著差异法（Least Significant Difference, LSD），该方法可以

判断到底是哪些均值之间存在差异。LSD 是通过对两个总体均值是否相等的 t- 检验方法，对总体方差估计用 MSE 来代替修正所得。LSD 方法可以控制 I 型比较错误率（Comparison-Wise Error Rate， CER），比较灵敏。但 SNK 方法可以控制试验错误率 EER。

```
程序21-14  单因子方差分析- LSD方法
proc glm data=mydataG;
  class factor;
  model x=factor ;
  means factor /hovtest lsd; /*使用 LSD 方法*/
run;
quit;
```

程序 21-14 运行后的输出结果如图 21-14 所示。



单因素方差分析

GLM 过程

t Tests (LSD) for x

Note: This test controls the Type I comparisonwise error rate, not the experimentwise error rate

Alpha	0.05
Error Degrees of Freedom	19
Error Mean Square	19.05263
Critical Value of t	2.09302

Comparisons significant at the 0.05 level are indicated by ***.

factor 比较	均值间差值	95% Confidence Limits		
A4 - A2	10.000	4.468	15.532	***
A4 - A1	15.000	9.651	20.349	***
A4 - A3	16.000	10.222	21.778	***
A2 - A4	10.000	-15.532	-4.468	***
A2 - A1	5.000	-0.083	10.083	
A2 - A3	6.000	0.468	11.532	***
A1 - A4	15.000	-20.349	-9.651	***
A1 - A2	-5.000	-10.083	0.083	
A1 - A3	1.000	-4.349	6.349	
A3 - A4	16.000	-21.778	-10.222	***
A3 - A2	6.000	11.532	-0.468	***
A3 - A1	1.000	-6.349	4.349	

图 21-14　变量 x 的 t- 检验 (LSD)

从图 21-14 中可以看到，A_4 和 A_1、A_2、A_3 差异非常显著，同时 A_2 和 A_3 之间差异也极其显著。LSD 方法的结果与对应 SNK 的方法结果（见图 21-15）大同小异；SNK 方法是排好序且更加简洁，比如 A_4 和 A_1 差值从图 21-15 中一眼就可以看出 66-51 15，但 LSD 提供了更加详细的信息。

具有相同字母的均值稍有不同

SNK 分组	均值	N	factor
A	66.000	5	A4
B	56.000	6	A2
B			
B	51.000	7	A1
B			
B	50.000	5	A3

图 21-15　变量 x 的 SNK 检验

单因子方差分析是方差分析的基础，因此我们了解单因子方差分析的基本原理和工作方法对掌握方差分析具有重要意义。双因子方差分析继承了它的基本概念，其中无交互作用的双因子方差分析与单因子方法分析基本相同。

21.2.3 双因子方差分析

有交互作用的双因子方差分析除了主效应外还有双因子联合效应，因此其总变异量可以分解为4部分：两个因子各自的主效应，两个因子交互作用效应，以及各因子的组内变异量。相应地F检验也就变为3个检验：双因子各自的F检验，以及交互作用的F检验。就是因子A组间变异、因子B组间变异以及因子AB交互作用变异3个均方差与组内变异的比值。当交互作用不显著时，可以将两个因子各自的F检验结果当作双因子方差分析的结论。

比如某彩电生产商研发了4款彩电品牌（A_1~A_4），在中国5个地区（B_1~B_5）进行销售，其销售量如表21-4所列。现在要分析彩电品牌和销售地区是否对彩电的销量有显著影响。

表21-4 5个地区的四款彩电销售

A \ B	B_1	B_2	B_3	B_4	B_5
A_1	365	350	343	340	323
A_2	345	368	363	330	333
A_3	358	323	353	343	308
A_4	288	280	298	260	298

我们分别对因子A和B作零假设：品牌对彩电销售量没有影响，地区对彩电销售量没有影响。程序21-15的代码中我们用factor1和factor2分别表示因子A和B：

程序21-15 双因子方差分析

```
data mydata;
  do factor1 = 1 to 4;
    do factor2 = 1 to 5;
      input x @@;
      output;
    end;
  end;
cards;
365 350 343 340 323
345 368 363 330 333
358 323 353 343 308
288 280 298 260 298
run;

proc glm data=mydata;
  class factor1 factor2;
  model x=factor1 factor2;
  means factor1 factor2/duncan alpha=0.01;
  run;
quit;
```

运行结果如图21-16所示。其中我们可以看到因子factor1和factor2的F值分别为

18.11 和 2.10。分别查对应的自由度的 $F_{0.01}$ 临界值 $F_{0.01}(3, 12)=3.49$，$F_{0.01}(4, 12)=3.26$。对于因子 factor1，由于 18.11 > 3.49，拒绝零假设，说明品牌对彩电的销售量具有显著的影响。而对于因子 factor2，由于 2.10 < 3.26，接纳零假设，说明销售地区对彩电的销售量没有显著影响，各地区的销售情况没有什么差别。

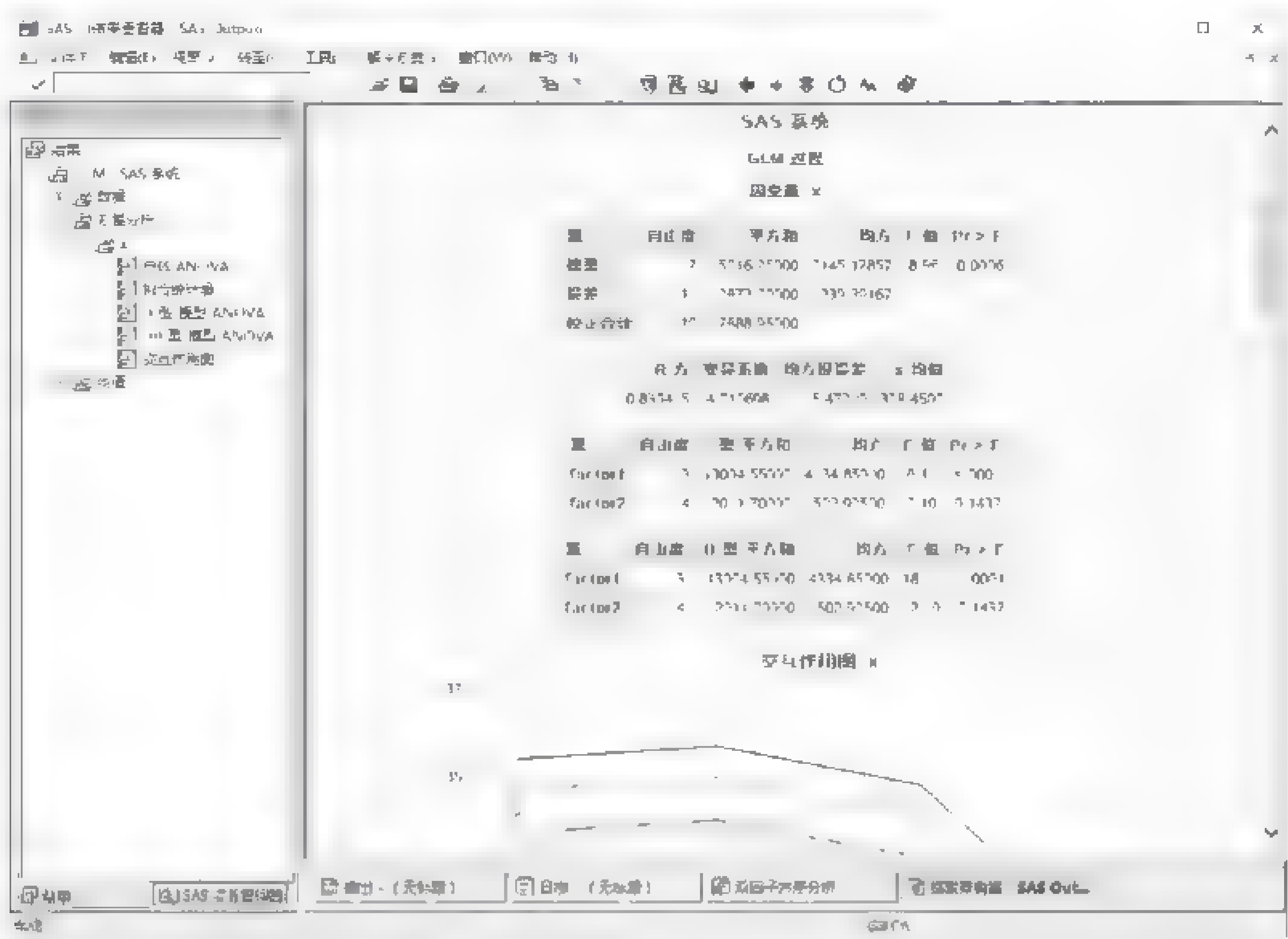
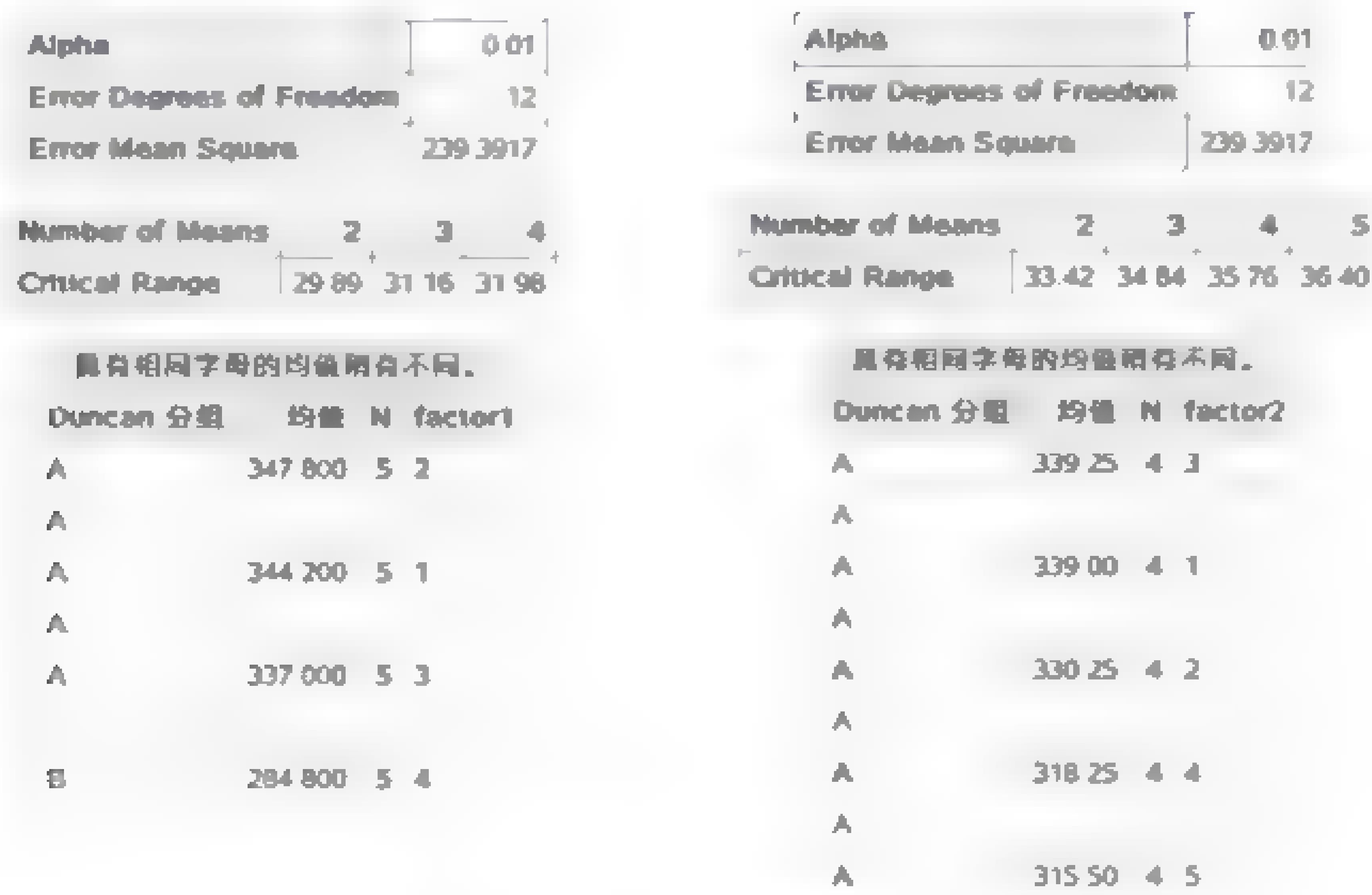


图 21-16 双因子方差分析

我们进一步检查因子 factor1 和 factor2 的多重因子比较结果如下，可知 factor1 中的第 4 个品牌与其他 3 个品牌具有显著不同，说明该品牌的销售量明显拖后腿。而 factor2 为销售地区，我们其实已经知道它对彩电销售量没有什么影响，它们也都归入相同的 Duncan 分组之中（见图 21-17）。



Alpha	0.01
Error Degrees of Freedom	12
Error Mean Square	239.3917

Number of Means	2	3	4
Critical Range	29.89	31.16	31.98

具有相同字母的均值明显不同。

Duncan 分组	均值	N	factor1
A	347.800	5	2
A			
A	344.200	5	1
A			
A	337.000	5	3
B	284.800	5	4

Alpha	0.01
Error Degrees of Freedom	12
Error Mean Square	239.3917

Number of Means	2	3	4	5
Critical Range	33.42	34.84	35.76	36.40

具有相同字母的均值明显不同。

Duncan 分组	均值	N	factor2
A	339.25	4	3
A			
A	339.00	4	1
A			
A	330.25	4	2
A			
A	318.25	4	4
A			
A	315.50	4	5

图 21-17 双因子方差分析多重因子比较

如果因子之间存在交互作用，SAS 分析代码和前例没有显著不同，只需要将交互作用的因子 factor1*factor2 在模型中给出即可（见程序 21-16）。

程序21-16　有交互作用的双因子方差分析

```
proc glm data=mydata;
  class factor1 factor2;
  model x=factor1 factor2 factor1*factor2;
  means factor1 factor2/duncan alpha=0.01;
run;
quit;
```

需要特别注意的是，进行存在交互作用的双因子分析，要求数据必须有重复的，否则是不能检验两个因子的交互作用是否显著。

第22章

数据标准化

在对多变量数据进行统计分析处理时，往往需要对变量数据进行统一尺度或者无量纲化，称为数据标准化。数据标准化就是将数据按比例缩放使之投影到一个特定区间，为指标处理的比较和评价消除单位影响，使不同单位或者量级的变量能够进行比较、累加和汇总运算。

典型的数据标准化包括数据归一化处理，就是将数据映射到 [0, 1] 区间上，它不但能提高模型的收敛速度，也能提升模型的精度。比如聚类分析中对象之间距离的计算，归一化可抑制指标值较大的变量对结果的影响，保证各特征对结果贡献相同。消除量纲则主要解决数据的可比性，使它们在对模型的贡献上具有平衡的影响。另外，除非各维度数据分布范围本来就比较接近，有些模型在各维度不均匀伸缩后，最优解与原来数据计算出来的解不等价（如 SVM），这种情况下则必须对数据进行标准化才能进一步处理。而一些模型不管伸缩与否，所求的最优解与数据变换前等价（如逻辑回归），则标准化只会改善收敛速度，此时标准化处理是可选的。

在数据分析科学中有相当多的统计量或指标取值在区间 [-1, 1] 上，我们也一般倾向于通过取绝对值或者平方将数据变换到 [0, 1] 上，这样就跟概率 P 值的取值范围相同。

22.1　常用标准化方法

数据标准化的方法非常多，它本质上就是一种数据变换。下面列出了一些常用的方法：

（1）Min-Max 标准化：也称离差标准化或 0-1 标准化，它将原始数据按照 [min, max] 线性映射到 [0, 1] 上。由于此种标准化方法依赖于最大值和最小值构成的极差，因此变换比较不稳定，但它是最简单的数据标准化方法：

$$X' = \frac{(X - \min)}{(\max - \min)}$$

如果希望将数据映射到 [-1, 1] 上，可以将原始数据减去均值再除以极差：

$$X' = \frac{(X - \mu)}{(\max - \min)}$$

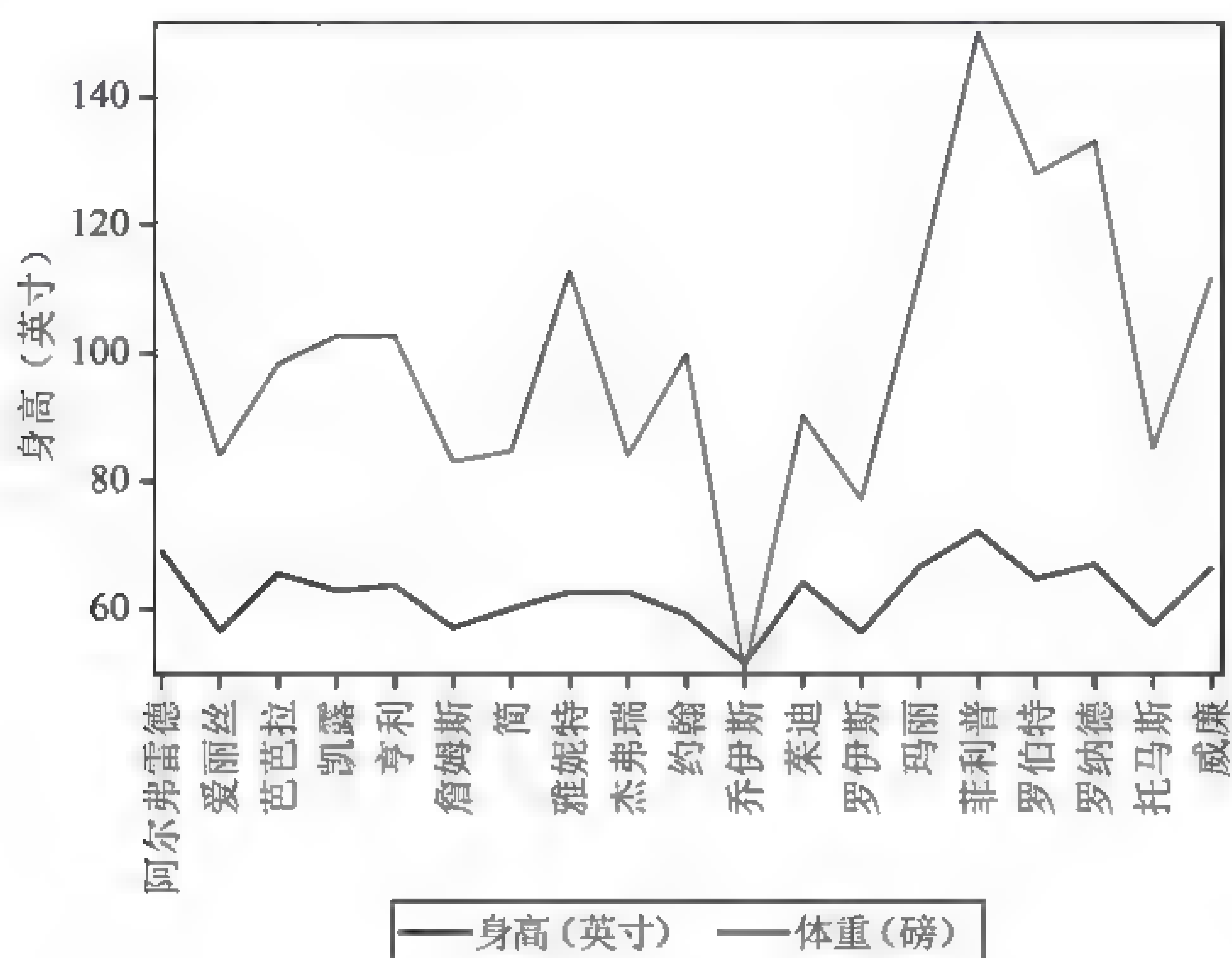
（2）Z-Score 标准化：是最常用的标准化方法，它将原始数据变换为均值为 0 和方差为 1 的新数据。如果原始数据符合正态分布，即 $X \sim N(\mu, \sigma^2)$，则标准化数据符合标准正态分布 $X \sim N(0, 1)$。这种标准化方法适用于总体数据的最小值 / 最大值未知，或样本数据不包含离群值的情况。计算方法就是对每列数据计算均值 μ 和标准差 σ，然后将原始

数据减去均值 μ，再除以标准差 σ。

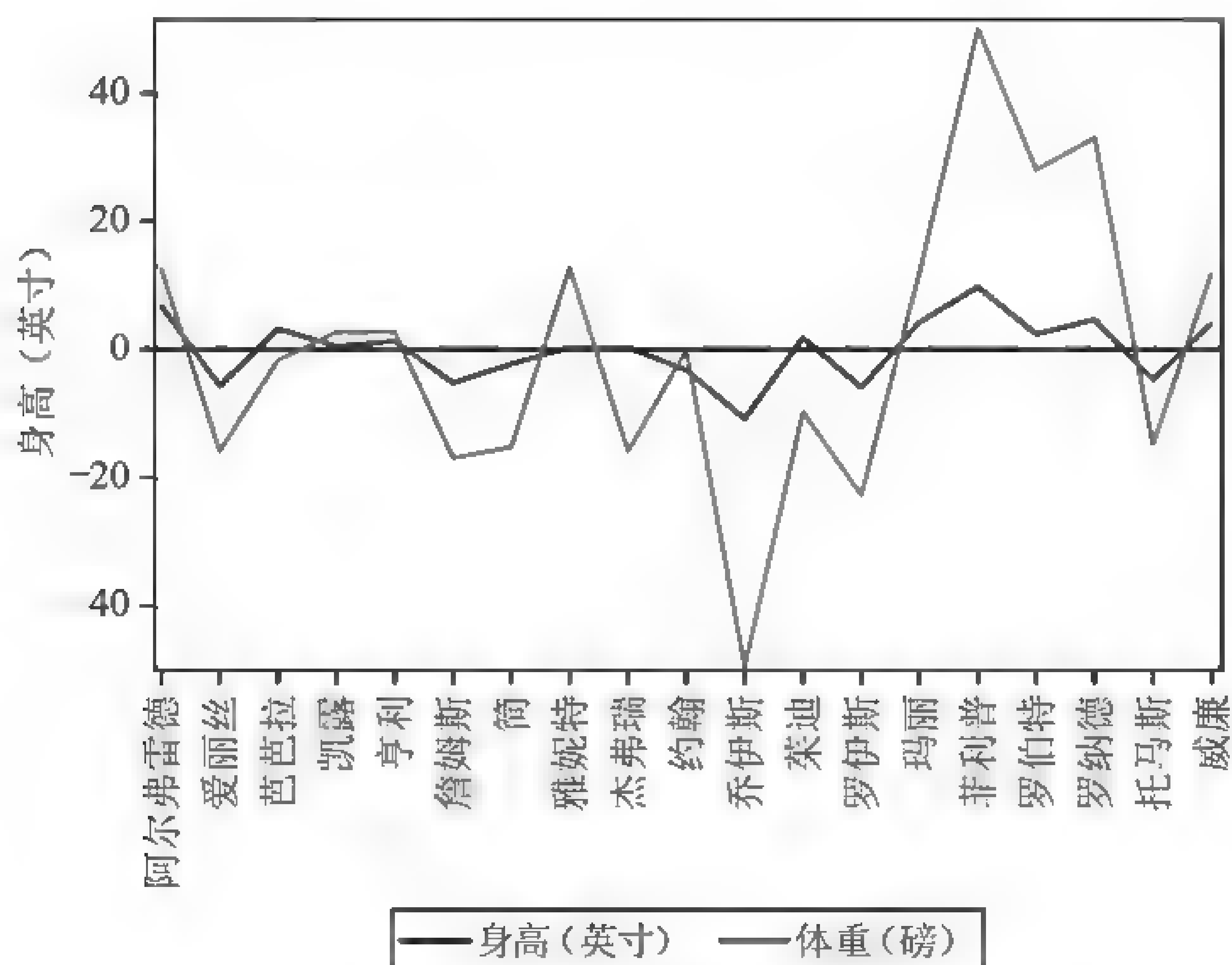
$$X' = \frac{X - \mu}{\sigma}$$

标准化后的数据如果大于 0 表示高于平均水平，否则表示低于平均水平。Z-Score 标准化后的数据并不在落在 [0, 1] 区间上，而是标准化后数据的均值为 0，方差为 1。

图 22-1 展示了 SAS 系统数据 SASHELP.CLASS 两个变量身高和体重，使用 Z-Score 标准化前和标准化后的对比图。



（a）



（b）

图 22-1　数据 Z-Score 标准化前后

（3）小数定标标准化：按照原始数据中绝对值最大的那个数值，将所有数据的小数点移动指定位数来缩放数据到 [-1, 1] 区间上：

$$X' = \frac{X}{10^j}$$

其中 $j = \mathrm{ceil}(\log_{10}^{\max(|X|)})$。$j$ 是绝对值的最大值取常用对数后，向上取整所得到的整数。例如数据 12, 345, 678, −987, −654, −321，其绝对值的最大值为 987，取常用对数变换后向上取整结果为 3，因此小数点需要向前移动 3 位，则原始数据都除以 10^3 得到 0.012, 0.345, 0.678, −0.987, −0.654, −0.321。

(4) 最大绝对值归一标准化：将绝对值最大的那个值归一，所有的数据除以它将数据变换到 [-1, 1] 之间。

$$X' = \frac{X}{\max(|X|)}$$

(5) 总和归一标准化：如果指标数据都大于零，可将所有指标数据总和归一，所有数据除以它变换到 [0, 1] 之间。因此，该变换会将整个变量的总和压缩到 1。

$$X' = \frac{X}{\mathrm{Sum}(x)}$$

（6）Logistic 幂函数变换：使用幂函数将原始数据两极化，该变换使得负无穷到 0 之间的数据趋向于 0，大于零到正无穷的数据趋向于 1。这种变换通常称为 S 函数变换，广泛用于神经网络和逻辑回归分析中（见图 22-2）。

$$X' = \frac{1}{1+e^{-X}}$$

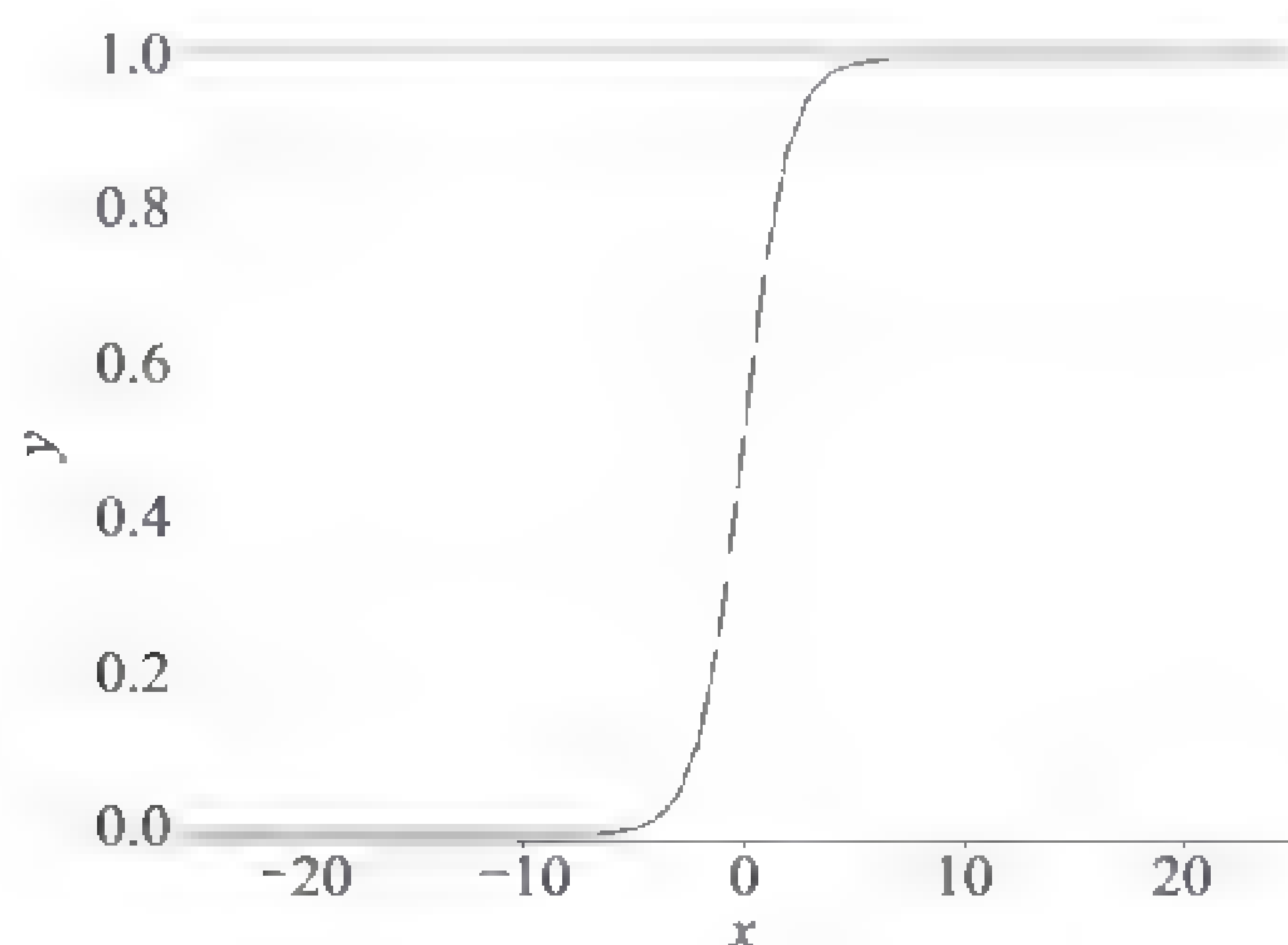


图 22-2　Logistic 幂函数变换

（7）atan 三角函数变换：该变换可以将数据变换到 [−1, 1] 之间，其中大于零的数据被映射到 [0,1] 区间上，而小于零的数据被映射到 [−1, 0] 区间上。与上面的 Logistic 幂函数变换有类似之处（见图 22-3）。

$$X' = \frac{2}{\pi}\mathrm{atan}(X)$$

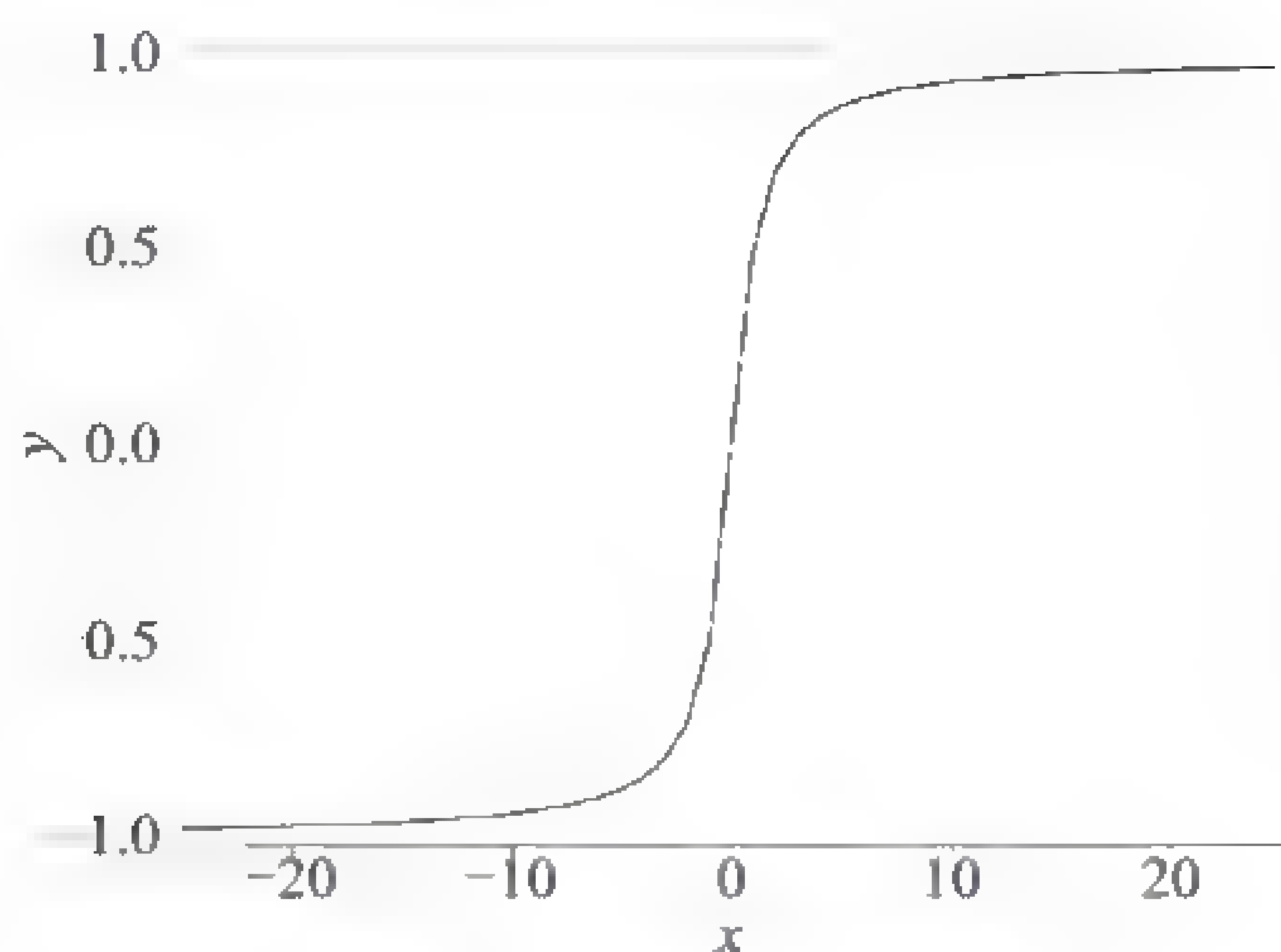


图 22-3　Atan 三角函数变换

（8）Log10 对数函数变换：当所有的数据都大于等于 0 时，可以使用常用对数函数进行变换（见图 22-4 所示）。即

$$X' = \log 10(X)$$

如果希望变换后的数据落到 [0, 1] 区间上，还需要除以最大值的对数变换值，即

$$X' = \frac{\log 10(X)}{\log 10\left[\max\left(|X|\right)\right]}$$

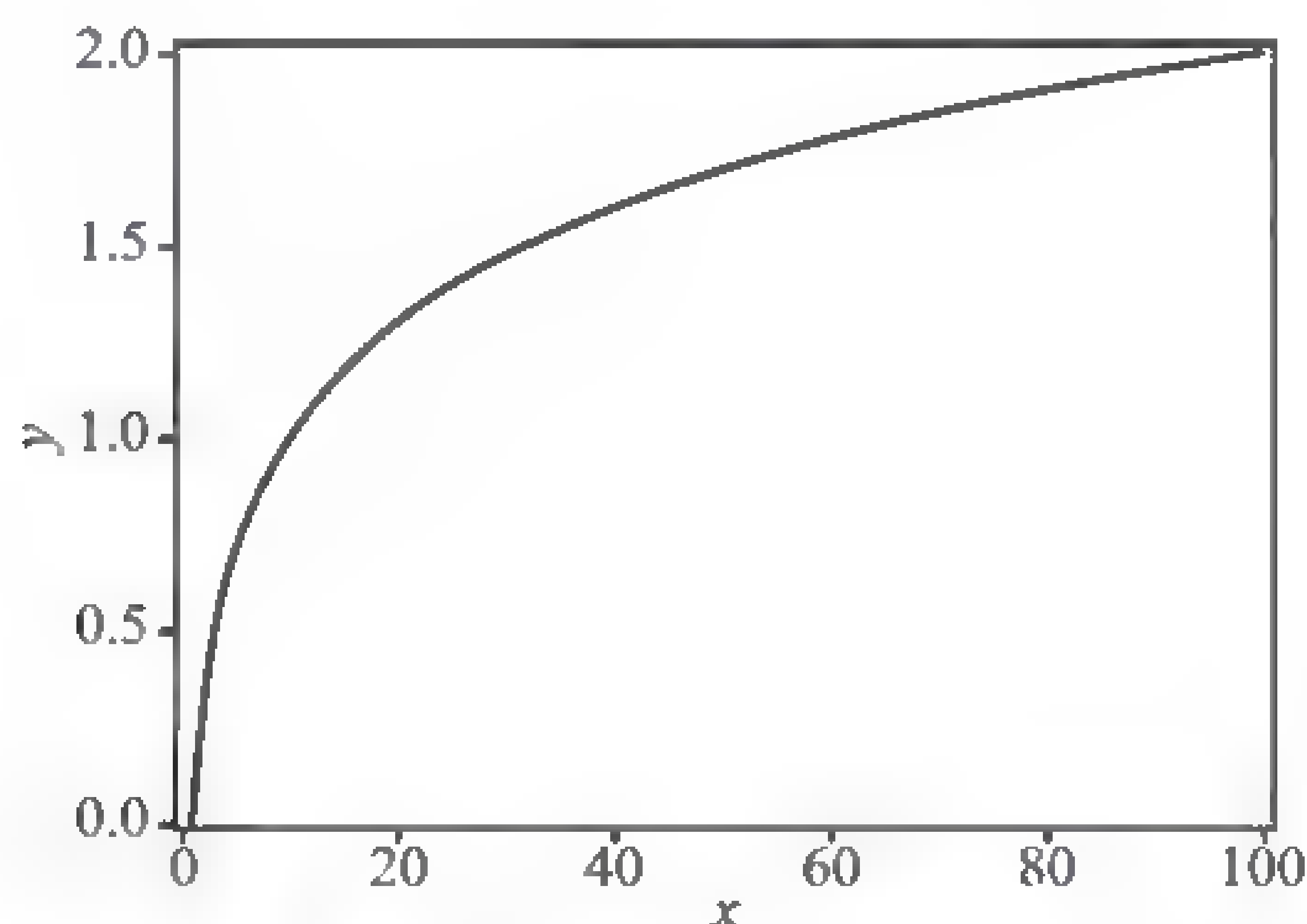


图 22-4　Log10 常用对数变换

22.2　SAS 数据标准化

在 SAS 中数据标准化可采用多种方法，甚至有些数据分析过程步自带选项，对数据进行自动标准化处理。SAS 中专门处理数据标准化的过程步有 PROC STANDARD 和 PROC STDIZE，如果用户需要扩展数据标准化方法，可自己编写函数实现。

(1) 使用过程步 PROC STANDARD 来将原始数据标准化需要指定均值 mean 和标准差 std。其本质就是上面的 Z-Score 标准化方法。数据标准化只能作用在量化变量如 age、

height 和 weight 上（见程序 22-1）。

程序22-1 z-Score 数据标准化示例

```
proc means data=sashelp.class;
run;

/*调用 PROC STANDARD 进行标准化*/
proc standard data=sashelp.class mean=0 std=1 out=class_std;
  var age height weight;
run;
/*验证标准化后数据的均值和标准差是否为 0,1*/
proc means data=class_std;
run;
```

系统输出如图 22-5 所示，从 PROC MEANS 的输出中可以看到，各列数据被标准化成均值为 0，标准差为 1 的数据。

变量	标签	N	均值	标准差	最小值	最大值
Age	年龄	19	13.3157895	1.4926722	11.0000000	16.0000000
Height	身高（英寸）	19	62.3368421	5.1270752	51.3000000	72.0000000
Weight	体重（磅）	19	100.0263158	22.7739335	50.5000000	150.0000000

变量	标签	N	均值	标准差	最小值	最大值
Age	年龄	19	-4.67462E-17	1.0000000	-1.5514388	1.7982586
Height	身高（英寸）	19	9.582978E-16	1.0000000	-2.1526585	1.8847310
Weight	体重（磅）	19	-5.25895E-17	1.0000000	-2.1746931	2.1943370

图 22-5 标准化前后的基本统计量

如果希望在标准化后的数据集中保留变换之前的数据，可以采用如下两种方法实现。

将需要标准化的变量或数据列复制一份，然后仅对这些新列进行标准化。程序 22-2 的代码中先创建了 3 个计算变量 AGE_STD、HEIGHT_STD 和 WEIGHT_STD ，然后对新建列作标准化。

程序22-2 标准化后保留原有的数据列：临时数据集方法

```
data class_tmp;
  set sashelp.class;
  age_std    =age;
  height_std =height;
  weight_std =weight;
run;

proc standard data=class_tmp mean=0 std=1 out=class_std;
  var age_std height_std weight_std;
run;
proc print data=class_std;run;
```

先对数据进行标准化，然后用 PROC SQL 将标准化之前的数据合并到结果数据集中，见程序 22-3 所示：

程序22-3 标准化后保留原有的数据列：PROC SQL 合并

```
proc standard data=sashelp.class mean=0 std=1 out=class_tmp;
  var age  height  weight;
run;
proc sql;
  create table class_std as
  select orig.name,
    orig.sex,
    orig.age,
    orig.height,
    orig.weight,
    tmp.age as age_std,
    tmp.height as height_std,
    tmp.weight as weight_std
  from sashelp.class as orig,  work.class_tmp as tmp
  where orig.name=tmp.name;
run;
quit;
```

（2）SAS 专业统计包 SAS/STAT 提供过程步 PROC STDIZE，它提供多达 19 种数据标准化方法，默认方法为 STD，即 Z-Score 标准化。每种标准化方法以数据分布的位置和尺度衡量，变换时将位置 L 对应的统计量变换为 0，再除以尺度 S，即 $X'=(X-L)/S$。表 22-1 列出了 SAS 支持的数据标准化方法名称以及对应的位置 L 和尺度 S。

表 22-1 PROC STDIZE 支持的数据标准化方法

方法 Method	位置 L	尺度 S	说　明
RANGE	最小值	极差	Min-Max 标准化
MIDRANGE	极差的中点	半极差	
MEDIAN	中位数	1	
IQR	中位数	四分位距	标准化后 Median=0
MAD	中位数	中位数绝对偏差	
MEAN	均值	1	
AGK(p)	均值	AGK 估计 (ACECLUS)	
STD	均值	标准差	Z-Score 标准化
SUM	0	和	总和归一法
MAXABS	0	绝对值的最大值	最大绝对值归一法
EUCLEN	0	欧氏距离长度	
USTD	0	原始标准差	
SPACING(p)	中间平均值	最小间距	
AWAVE(c)	Wave 单步 M- 估计	Wave A- 估计	
AHUBER(c)	Huber 单步 M- 估计	Huber A- 估计	
ABW(c)	双权 单步 M- 估计	双权 A- 估计	
IN(ds)	从数据读入	从数据读入	
L(p)	L(p)	L(p)	

比如用 Z-Score 标准化方法标准化数据的 METHOD 参数为 STD，它与 PROC STANDARD 等价，其他常用的标准化方法也一并列举如程序 22-4 所示。

程序22-4 常用数据标准化方法在PROC STDIZE 中的实现

```
proc stdize data=sashelp.class out=class_std method=STD;
  var age height weight;
run;

/*Min-Max 标准化：method=RANGE*/
proc stdize data=sashelp.class out=class_std method=RANGE;
  var age height weight;
run;

/*总和归一标准化：method=SUM*/
proc stdize data=sashelp.class out=class_std method=SUM;
  var age height weight;
run;

/*最大绝对值归一标准化：method=MAXABS*/
proc stdize data=sashelp.class out=class_std method=MAXABS;
  var age height weight;
run;
```

（3）由于大部分数值计算是基于二维数组进行的，能否重用SAS过程步STANDARD或STDIZE对该二维数组进行标准化呢？答案是肯定的。可以先将PROC步封装成SAS宏%standard，然后再通过FCMP函数调用该SAS宏来实现。这种方法可实现在一个DATA步内调用另外一个PROC步的功能（见程序22-5）。

程序22-5 将PROC STANDARD 过程步封装为SAS 宏

```
/*将 PROC STANDARD 封装为宏*/
%macro standard;
  %let dsname=%sysfunc(dequote(&DSNAME));
  proc standard data=&DSNAME mean=&MEAN std=&STD out=&dsname;
  run;
%mend standard;
```

此时该SAS宏可以在数据集的层面上进行调用（如程序22-6），但我们的目的是在数组上使用该PROC STANDARD过程步。因此，还需要利用FCMP实现桥接。

程序22-6 %STANDARD 宏示例

```
data class2;
  set sashelp.class;
run;
%let dsname=class2;
%let mean=0;
%let std=1;
%standard;
```

程序22-7利用FCMP函数将DATA步的程序空间和PROC步的程序空间进行桥接：

程序22-7 将SAS 宏封装为 FCMP 函数以对数组元素进行标准化处理

```
proc fcmp outlib=work.funcs.standard;
  function standard(x[*,*], mean, std);
    outargs x;
    dsname='work._TMP_';
    rc=write_array(dsname, x); /*将数组写到临时表*/
    rc=run_macro('standard', dsname, mean, std);/*用宏对临时表标准化*/

    if rc=0 then do;
      array y[1] _temporary_; /*需要将单列和多列分别处理*/
```

```
      rows=dim(x); cols=dim(x,2);

      if cols=1 then do;
        call dynamic_array(y, rows);
        rc=read_array(dsname, y);
        do i=1 to rows;
          x[i,1]=y[i ];
        end;
      end;
      else do;
        call dynamic_array(y, rows, cols);
        rc=read_array(dsname, y);
        do i=1 to rows;
          do j=1 to cols;
            x[i,j]=y[i,j];
          end;
        end;
      end;
    end;
    return(rc);
  endsub;
run;
```

这样就可以在 DATA 步内调用 FCMP 函数对内存中的数组进行数据标准化了（见程序 22-8）。

程序22-8　对变量数组进行数据标准化

```
options cmplib=(work.funcs);
data class_std ;
  array w[19,3] _temporary_;
  do i=1 to rows;
    set sashelp.class point=i nobs=rows;
    w[i,1]=age;
    w[i,2]=height;
    w[i,3]=weight;
  end;

  rc=Standard (w, 0, 1);

  if rc=0 then do;
    do i= 1 to rows;
      set sashelp.class point=i nobs=rows;
      age=w[i,1];
      height=w[i,2];
      weight=w[i,3];
      output;
    end;
  end;
  drop rc;
  stop;
run;
proc print data=class_std;run;
```

系统输出 class_std 数据集，其中 Weight 和 Height 列数据被调用 PROC STANDARD 标准化。为方便调用，一般用二维数组做参数，这样就可以处理多列。

22.3　自定义数据标准化

如果 SAS 提供的各种数据标准化方法不能满足业务需求，用户必须自己实现特定的数据标准化方法以及数据变换操作。实践中由于数据变换一般面向数组进行，则我们需要定义通用的函数对输入数组参数进行处理。下面以 FCMP 函数实现小数定标标准化和 Z-Score 标准化为例加以说明（见程序 22-9）。

程序22-9　自定义实现小数定标标准化方法

```
proc fcmp outlib=work.funcs.preprocess library=work.funcs;
  /*求二维数组某列的最小值*/
  function getcolmin(d[*,*],col);
    cols=dim(d,2);
    if col <= cols then do;
      rows=dim(d,1);
      colmin=d[1,col];
      do i=2 to rows;
        if d[i,col]<colmin then colmin=d[i,col];
      end;
      return(colmin);
    end;
    return (.);
  endsub;
  /*求二维数组某列的最大值*/
  function getcolmax(d[*,*],col);
    cols=dim(d,2);
    if col <= cols then do;
      rows=dim(d,1);
      colmax=d[1,col];
      do i=2 to rows;
        if d[i,col]>colmax then colmax=d[i,col];
      end;
      return(colmax);
    end;
    return (.);
  endsub;
  /*实现小数定标标准化方法 DecimalScaling*/
  function DecimalScaling(d[*,*],o[*,*]);
    outargs o;
    rows=dim(d,1);
    cols=dim(d,2);
    if rows=. or cols=. then return(1);
    do col=1 to cols;
      colmax=getcolmax(d, col);
      colmin=getcolmin(d, col);
      absmax=max( abs(colmax), abs(colmin));
      j=ceil(log10(absmax));
      do row=1 to rows;
        o[row,col]= d[row,col] /(10**j);
      end;
    end;
    return (0);
  endsub;
run;
quit;
```

然后用上面定义函数 DECIMALSCALING 来标准化 SASHELP.CLASS 数据集中的

数值列。其中需要将数值列 AGE、HEIGHT、WEIGHT 用 SAS 的随机访问机制读入到二维数组中，然后统一对各列进行变换（见程序 22-10）。

程序22-10 自定义小数定标标准化方法应用示例

```
options cmplib=(work.funcs);
data class_std ;
  array w[19,3] _temporary_ ;
  array o[19,3] _temporary_ ;

  do i=1 to rows;
    set sashelp.class point=i nobs=rows;
    w[i,1]=age;
    w[i,2]=height;
    w[i,3]=weight;
  end;

  rc=DecimalScaling (w, o);
  if rc=0 then do;
    do i=1 to rows;
      set sashelp.class point=i nobs=rows;
      age=o[i,1];
      height=o[i,2];
      weight=o[i,3];
      output;
    end;
  end;
  drop rc;
  stop;
run;
proc print data=class_std;run;
```

系统输出如图 22-6 所示，其中可以看到所有数值列都变成小数。

Obs	age	height	weight	Name	Sex
1	0.14	0.690	0.1125	阿尔弗雷德	男
2	0.14	0.635	0.1025	亨利	男
3	0.12	0.573	0.0830	詹姆斯	男
17	0.14	0.643	0.0900	茱迪	女
18	0.12	0.563	0.0770	罗伊斯	女
19	0.15	0.665	0.1120	玛丽	女

图 22-6 编程实现数据标准化

用户可用同样的方法实现最大绝对值归一标准化、总和归一标准化以及 Logistic 变换、ATAN 变换和 Log10 对数函数变换等。对于简单的变换如 Logistic 变换、ATAN 变换和 Log10 变换可在 DATA 步内实现如下计算列即可完成，但对于需要遍历所有样本的数据标准化方法，一般还是使用封装函数进行比较妥当。

```
y=1 /(1 + exp( -x ));

pi=constant("PI");
y=atan( x ) * 2 / pi;

y=log10(x);
```

程序 22-11 实现了 Z-Score 数据标准化方法，而对应的应用示例可参考程序 22-12。

程序22-11　自定义实现 Z-Score 标准化方法

```
proc fcmp outlib=work.funcs.standardize ;
  /*求二维数组某列均值*/
  function getcolavg(d[*,*],c);
    cols=dim(d,2);
    if c <= cols then do;
      rows=dim(d,1);
      sum=0;
      do i=1 to rows;
        sum =sum + d[i,c];
      end;
      return( sum / rows);
    end;
    return (.);
  endsub;
  /*求二维数组某列方差*/
  function getcolS2(d[*,*],c);
    cols=dim(d,2);
    if c <= cols then do;
      avg=getcolavg(d, c);
      rows=dim(d,1);
      sum=0;
      do i=1 to rows;
        sum =sum + (d[i,c]-avg)**2;
      end;
      return( sum / (rows-1));
    end;
    return (.);
  endsub;
  /*自己实现 Z-Score 标准化*/
  function Standardize(d[*,*], o[*,*]);
    outargs o;
    rows=dim(d,1);
    cols=dim(d,2);
    if rows^=dim(o,1) or cols^=dim(o,2) then return(1);

    array avg[1] /nosymbols;
    call dynamic_array(avg, cols);
    array s[1] /nosymbols;
    call dynamic_array(s, cols);
    do c=1 to cols;
      avg[c]=getcolavg(d,c);
      s2=getcolS2(d,c);
      if s2=0 then return(1);/*标准离差小于等于0，不能标准化处理*/
      s[c]=sqrt( s2 );
    end;
    do r=1 to rows;
      do c=1 to cols;
      o[r,c]=(d[r,c]-avg[c])/s[c];
    end;
  end;
  return (0);
  endsub;
run;
quit;
```

程序22-12 自定义z-Score 数据标准化应用示例，结果输出到 class_std 数据集中

```
options cmplib=(work.funcs);
data class std ;
  array w[19,3]  temporary ;
  array o[19,3]  temporary ;

  do i=1 to rows;
    set sashelp.class point=i nobs=rows;
    w[i,1]=age;
    w[i,2]=height;
    w[i,3]=weight;
  end;

  rc=Standardize (w, o);
  if rc=0 then do;
    do i=1 to rows;
      set sashelp.class point=i nobs=rows;
      age=o[i,1];
      height=o[i,2];
      weight=o[i,3];
      output;
    end;
  end;
  drop rc;
  stop;
run;
```

如果用 PROC 完成数据标准化，其等价的 SAS 代码为

```
proc standard data=sashelp.class out=class_std mean=0 std=1;run;
```

第23章

主成分分析与因子分析

当从多个属性或特征考察对象时，得到的观测数据就包含多个变量，此时待分析的数据也就包括多个变量，摆在数据科学家面前的首要任务是分析研究不同变量之间的关系。比如分析各变量之间是否存在统计学上非确定性的相关关系，包括平行关系和依存关系。处理多个变量数据的统计方法统称为多元统计方法，主要包括相关分析（Corralation Analysis）、典型相关分析（Cannonial Cooraltion Analysis）、主成分分析（Principal Component Analysis）、因子分析（Factor Analysis）、聚类分析（Cluster Analysis）和判别分析（Discriminant Analysis）等。统计分析方法种类繁多，但分析的目的往往是对类别型变量进行分类，或是对数值型变量进行预测。笔者整理的常用单变量和多变量数据分析方法如图 23-1 所示。



图 23-1　常用数据分析方法一览表

在多元统计分析方法中，主成分分析与因子分析方法是用来分析一组变量中的各个变量之间关系的首要分析方法。当研究的目标是两组变量（两个变量集）之间的关系时，

我们需要使用典型相关分析，它通过在每组变量中寻找若干能反映两组变量间最大相关可能的变量之线性组合，对两组变量之间的关系进行汇总和解释。

23.1 主成分分析

主成分分析（Principal Component Analysis，PCA）是统计学家卡尔·皮尔逊在1901年对非随机变量的分析和数理建模时引入的，后来被广泛应用于随机变量领域。其核心思想是在尽可能保留样本中反映数据变异特征的方差，即统计学上数据所蕴含的信息量的前提下，如何找到比原来变量数目更少的变量线性组合，以降维思想来简化分析数据。这些变量的线性组合称为主成分（Principal Component）。主成分分析大致揭示了变量之间可能存在的线性依存关系，它不但可用来汇总数据，也可以用来对高维数据进行降维，减少后续回归和聚类等数据分析中需要处理的变量数目，从而提供了一种将数据从高维空间投影至低维空间的手段。其数学模型可概括为

$$F = U'X \text{ 即 } F_j = f(X_1, X_2, \cdots, X_p) \equiv F_j = \sum_{i=1}^{p} u_{ij} X_i \text{ 其中 } (i, j = 1, 2, \cdots, p, m < p)$$

根据测量尺度不同，对于类别型数据，可以使用原始类别数据或者它的频数数据作为输入，进行简单对应分析或多重对应分析（Correspondence Analysis），用来揭示各类别变量所代表的类别之间的关系。对应分析使用类别变量的列联表而非协方差矩阵进行运算，其本质是一种加权形式的主成分分析 。对应分析的目的是在少数几个维度上对若干类别变量之间的关联关系（Associations）进行汇总，对应分析的输出结果为各类别在维度上的投影图，投影图上的两个轴分别反映对应分析的维度，各维度所能解释总信息量的百分比一般不同。

在定性数据的主成分分析以及多维偏好分析中，有些类别数据虽然以定量数据表示，但通过对数据应用恰当的线性或非线性变换，可改善协方差矩阵或相关系数矩阵的属性。对于连续型数据则通常可使用标准的主成分分析方法进行分析，输出标准化或非标准化主成分得分。在主成分分析输出的成分评分投影图中，两个轴上反映的是主成分，而各主成分能够解释总体变异的百分比一般是不同的。

主成分分析可基于协方差矩阵（Covariance Matrix）进行也可基于相关系数矩阵（Correlation Matrix）进行。当原始观测数据量纲不同，或者各变量的数据量级差别很大时，应该将原始数据标准化成均值为0，方差为1的标准化数据，再计算协方差矩阵，而此时协方差矩阵等价于相关系数矩阵。

相关系数矩阵值域在 [-1, 1] 上，所以比起协方差矩阵元素的数值具有更好的可解释性，因为相关系数矩阵中元素的绝对值越靠近1表明两个变量的相关性越强，越靠近0则表明相关性越弱，而正负号则表明它们是正相关还是负相关。

主成分分析与因子分析都是蕴含降维思想的数据分析方法，下面分别深入剖析其原理和方法，随后以具体的例子讲述如何对分析结果进行解释。

23.1.1　主成分分析原理

假设原始观测数据包括 p 个变量，每一个变量的数据由 n 次观测组成，其中 n 就是样本大小。

$$\begin{gathered} x_{11}, x_{21}, \cdots, x_{p1} \\ x_{12}, x_{22}, \cdots, x_{p2} \\ \vdots \\ x_{1n}, x_{2n}, \cdots, x_{pn} \end{gathered}$$

由 p 个变量组成的数据样本可看作是由 p 个向量 $X_1, X_2, \cdots, X_p$ 组成的数据，每个向量对应于数据表中的每一列。

$$X_i = (x_{i1}, x_{i2}, \cdots, x_{in}) \text{ 其中 } (i = 1, 2, \cdots, p)$$

要计算 p 个变量中任意两个变量 X_i，X_j 之间的协方差矩阵，可根据如下公式算出：

$$\mathrm{Cov}(X_i, X_j) = \frac{1}{n-1}\sum_{l=1}^{n}(x_{il} - \overline{x}_i)(x_{jl} - \overline{x}_i) \quad \text{其中} \quad \overline{x}_i = \frac{1}{n}\sum_{l}^{n} x_{il} \, (l = 1, 2, \cdots, n)$$

而计算 p 个变量中任意两个变量 X_i，X_j 之间的相关系数矩阵，则可以根据如下公式算出：

$$r(X_i, X_j) = \frac{\mathrm{Cov}(X_i, X_j)}{\sqrt{\mathrm{Var}(X_i)\mathrm{Var}(X_j)}} = \frac{\sum_{l=1}^{n}(x_{il} - \overline{x}_i)(x_{jl} - \overline{x}_j)}{\sqrt{\sum_{l=1}^{n}(x_{il} - \overline{x}_i)^2}\sqrt{\sum_{l=1}^{n}(x_{jl} - \overline{x}_j)^2}}$$

标准化数据计算出来的协方差矩阵跟未标准化数据计算出来的相关系数矩阵是等价的，也就是说协方差 $\mathrm{Cov}(X_i, X_j)$ 除以变量 X_i 的标准差 $\sqrt{\mathrm{Var}(X_i)}$ 和 X_j 的标准差 $\sqrt{\mathrm{Var}(X_j)}$，能消除了两个变量 X_i，X_j 的量纲影响。可以说相关系数矩阵是一种数据标准化后的特殊协方差矩阵，标准化数据计算出来的标准差为 1，对应上述公式中的分母为 1。

主成分分析通过对协方差矩阵或相关系数矩阵可求解特征值 λ 和特征向量 U，主成分分析在数学上就是一个正交线性变换，其本质就是将数据从变量空间变换到一个新的正交坐标系中，并且使数据的任何投影的第一大方差落在第一主成分轴 F_1 上，第二大方差落在第二个主成分轴 F_2 上，依此类推（见图 23-2）。因此主成分具有 3 个特性：各个主成分彼此独立，即协方差 $\mathrm{Cov}(F_i, F_j)=0$；对任一主成分 F_j，它对应各变量系数的平方和 $\sum_{i=1}^{p} u_{ij}^2 = 1$；第 1 主成分到第 p 主成分的方差贡献依次递减。

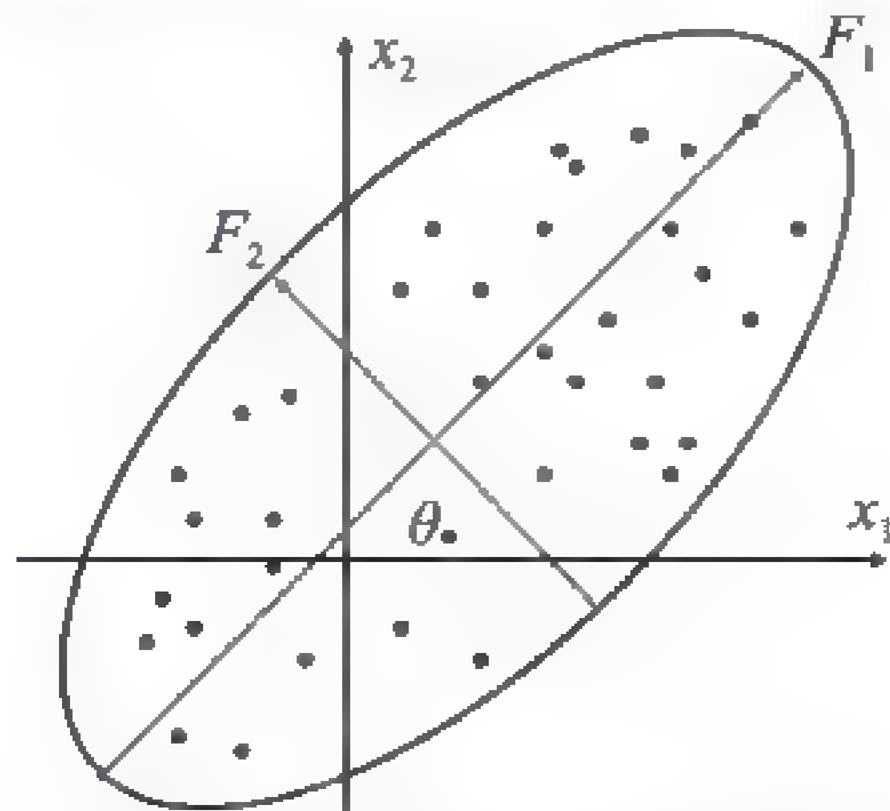


图 23-2　变量空间与主成分空间关系

主成分分析求解出的特征值λ_i之和恰好是原p个随机变量的方差之和，说明主成分分析将原来的p个随机变量的方差之和分解成了新的p个不相关的主成分方差贡献，每一个特征值λ_i就是主成分F_i对总体方差的贡献，因此$\lambda_i / \sum_{i=1}^{p}\lambda_i$被称为主成分$F_i$的贡献率。前$k$个主成分的累积方差贡献等于$\sum_{i=1}^{k}\lambda_i$，则它在全部方差中的累积贡献率为$\sum_{i=1}^{k}\lambda_i \Big/ \sum_{i=1}^{p}\lambda_i$。它说明这$k$个主成分能够反映原来$p$个变量所表示的信息量的百分比。实践中通常要求选入的k个主成分，其方差累计贡献率必须在80%以上。

构造出来的主成分和原变量之间满足关系$F=U'X$，U'为矩阵U的转置矩阵，矩阵$U=(u_1, u_2, \cdots, u_p)=[u_{ij}]$其中$i, j=1, 2, \cdots, p$。则第$j$个主成分$F_j$与原变量之间满足如下线性组合关系：

$$F_j = \sum_{i=1}^{p} u_{ij}X_i \quad \text{其中 } j=1,2,\cdots,m, n \leqslant p$$

而反过来，原变量与主成分之间则有$X=UF$，原变量与主成分的相关程度取决于对应的线性组合系数的大小：

$$\rho(X_i, F_j) = \mathrm{Cov}(X_i, F_j) \Big/ (\sigma_i\sqrt{\lambda_j}) = u_{ij}\lambda_j \Big/ (\sigma_i\sqrt{\lambda_j}) = u_{ij}\sqrt{\lambda_j} \Big/ \sigma_i$$

其中原变量X_i的方差为$\sigma_i^2 = \sum_{j=1}^{p} u_{ij}^{\ 2}\lambda_j$，$u_{ij}^{\ 2}\lambda_j$表示主成分$F_j$能解释的原始变量$X_i$的方差，如果提取了$m$个主成分，第$i$个原始变量$X_i$的信息被提取的比率为

$$\sum_{j=1}^{m} u_{ij}^{\ 2}\lambda_j \Big/ \sigma_i^2 = \sum_{j=1}^{m} \rho(X_i, F_j)^2$$

在SAS中有多个过程步能够做主成分分析，包括：

（1）PROC CORRESP 分类数据和频数数据的主成分分析，输入数据为列联表或原始数据，进行简单对应分析或多重对应分析。

（2）PROC PRINQUAL 使用于对定性数据执行主成分分析，包括类别数据和定序数据，也可用于多维偏好分析。

（3）PROC PRINCOMP 对连续型数据执行主成分分析，输出标准化或非标准化的主成分得分，是SAS中执行主成分分析的主要过程步。

（4）PROC FACTOR 执行各种探索性因子分析，支持旋转并输出主成分得分或因子得分的估计，需要注意该过程步默认选项执行的是主成分分析。

（5）PROC CANCORR 执行典型相关分析并输出典型变量得分。结果以表格展示，也可以将结果输出至数据集进行绘图。

（6）PROC PLS 使用包括偏最小二乘法在内的线性预测方法进行模型拟合，广泛应用于各种分析。PROC PLS 过程步也能执行主成分回归分析，回归的输出可用于预测。

23.1.2 主成分分析的具体步骤

下面以 SASHELP.CLASS 为例，用 PROC PRINCOMP 简单解释主成分分析的步骤

和逻辑。数据分析作为一种需要结合具体经验的解释型学科，分析方法本身对数据的语义并不敏感。因此才有“模型皆有误，或尤建奇功”这一至理名言。程序 23-1 和程序 23-2 分别基于相关系数和协方差矩阵对 SASHELP.CLASS 的量化变量进行主成分分析。

```
程序23-1　基于相关系数矩阵对 SASHELP.CLASS 做主成分分析
proc princomp data=sashelp.class ;
run;
```

或

```
程序23-2　基于协方差矩阵对 SASHELP.CLASS 做主成分分析
proc princomp data=sashelp.class COV ;
run;
```

基于协方差矩阵的主成分分析结果如下（见图 23-3），结果各部分的解释如下：

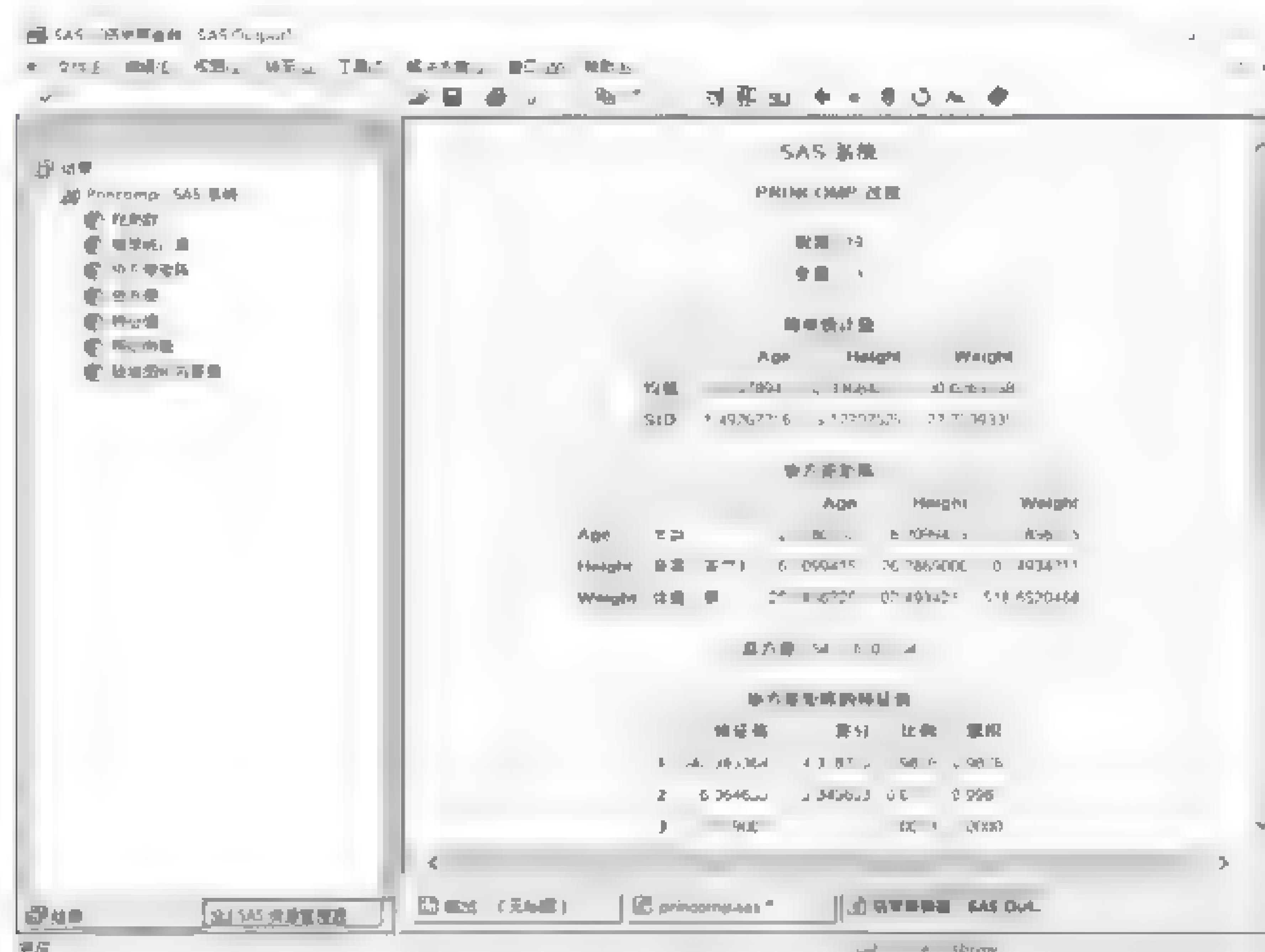


图 23-3　基于协方差矩阵的主成分分析结果

（1）观测数：其中可以解读出 PRINCOMP 使用 3 个定量变量 AGE、HEIGHT、WEIGHT 进行主成分分析，由于 PRINCOMP 是用于连续型变量的主成分分析，变量 NAME 和 SEX 为定类变量不参加分析，因此实际分析的观测数为 19，变量数为 3。

（2）简单统计量：包含各变量的均值和标准差，如 AGE 的均值为 13.31578947，标准差为 1.49267216，标准差的平方正是协方差矩阵（见图 23-4）对角线元素的值，如 AGE × AGE=2.2280702。

协方差矩阵

		Age	Height	Weight
Age	年龄	2.2280702	6.2099415	25.1856725
Height	身高（英寸）	6.2099415	26.2869006	102.4934211
Weight	体重（磅）	25.1856725	102.4934211	518.6520468

图 23-4　协方差矩阵

（3）协方差矩阵：列出了 3 个变量两两之间的协方差计算值，其中对角线元素为 3

个变量AGE、HEIGHT和WEIGHT各自的方差，如AGE × AGE = 2.2280702。

协方差对角线元素之和为3个原变量的总方差 $\sigma_1^2+\sigma_2^2+\sigma_3^2$，即547.16701754 = 2.2280702 + 26.2869006 + 518.5620468。

（4）总方差：列出了3个原始变量的总方差547.16701754，在数据分析科学中，方差代表了数据所隐含的信息量，类似于化学中描述系统混乱程度的熵值。

（5）特征值和特征向量：基于协方差矩阵可计算出3个特征值 λ_1 = 540.383364，λ_2 = 6.064653 和 λ_3 = 0.719001（见图23-5），分别代表主成分分析形成的3个主分量PRIN1、PRIN2和PRIN3对总体方差547.16701754的贡献，即 $\lambda_1+\lambda_2+\lambda_3$ = 总方差。

每一个主成分PRIN1、PRIN2和PRIN3对总体方差的贡献比率为 λ_i/总方差，它分别等于0.9876、0.0111和0.0013，累积贡献比率为0.9876、0.9987和1.0000。从中可以看出第一个分量PRIN1就已经贡献98.76%，超过了80%阈值，因此根据选取准则我们只需要一个主分量PRIN1就可以了。

协方差矩阵的特征值

	特征值	差分	比例	累积
1	540.383364	534.318710	0.9876	0.9876
2	6.064653	5.345653	0.0111	0.9987
3	0.719001		0.0013	1.0000

特征向量

		Prin1	Prin2	Prin3
Age	年龄	0.048098	0.220785	0.974136
Height	身高（英寸）	0.195851	0.954248	-0.225948
Weight	体重（磅）	0.979453	-0.201653	-0.002657

图23-5　特征值与特征向量

实际上，给定协方差矩阵或相关系数矩阵，在SAS中可以直接使用PROC IML过程步求解特征值和特征向量。比如上面的协方差矩阵，可用程序23-3的PROC IML过程步求解特征值和特征向量，其结果等价（见图23-6）。

程序23-3　PROC IML 计算特征值和特征向量

```
proc iml;
  A={
    2.2280702   6.2099415  25.1856725,
    6.2099415  26.2869006 102.4934211,
   25.1856725 102.4934211 518.6520468
  };
  call eigen(eigenvalues, eigenvectors, A);
  print A eigenvalues eigenvectors;
quit ;
```

A			eigenvalues	eigenvectors		
2.2280702	6.2099415	25.185673	540.38336	0.0480984	0.2207849	0.9741358
6.2099415	26.286901	102.49342	6.0646532	0.1958508	0.9542484	-0.225948
25.185673	102.49342	518.65205	0.7190007	0.9794534	-0.201653	-0.002657

图 23-6　PROC IML 计算特征值和特征向量

基于协方差矩阵的主成分分析结果，各主成分 PRIN1, PRIN2 和 PRIN3 是原变量 AGE, HEIGHT, WEIGHT 的线性组合。主分量和原变量之间的系数由输出的特征向量给出，此时主成分和原变量有如下数学关系：

```
Prin1=0.048098 * (Age - Age_Mean) + 0.195851 * (Height - Height_Mean)
    + 0.979453 * (Weight - Weight_Mean)
```

比如对于第 1 个观测，其主成分得分 Prin1 可由如下公式计算出：

```
Prin1=0.048098 * (14 - 13.31578947) + 0.195851 * (69 - 62.33684211)
    + 0.979453 * (112.5 - 100.0263158)=13.5553
```

（6）陡坡图和方差图：各特征值 λ_i 和主成分能够解释的方差可由陡坡图和方差图（见图 23-7）给出，其中方差图中上面的虚线为累计方差贡献，下面的实线为各分量的方差贡献，形状与陡坡图是一样的。

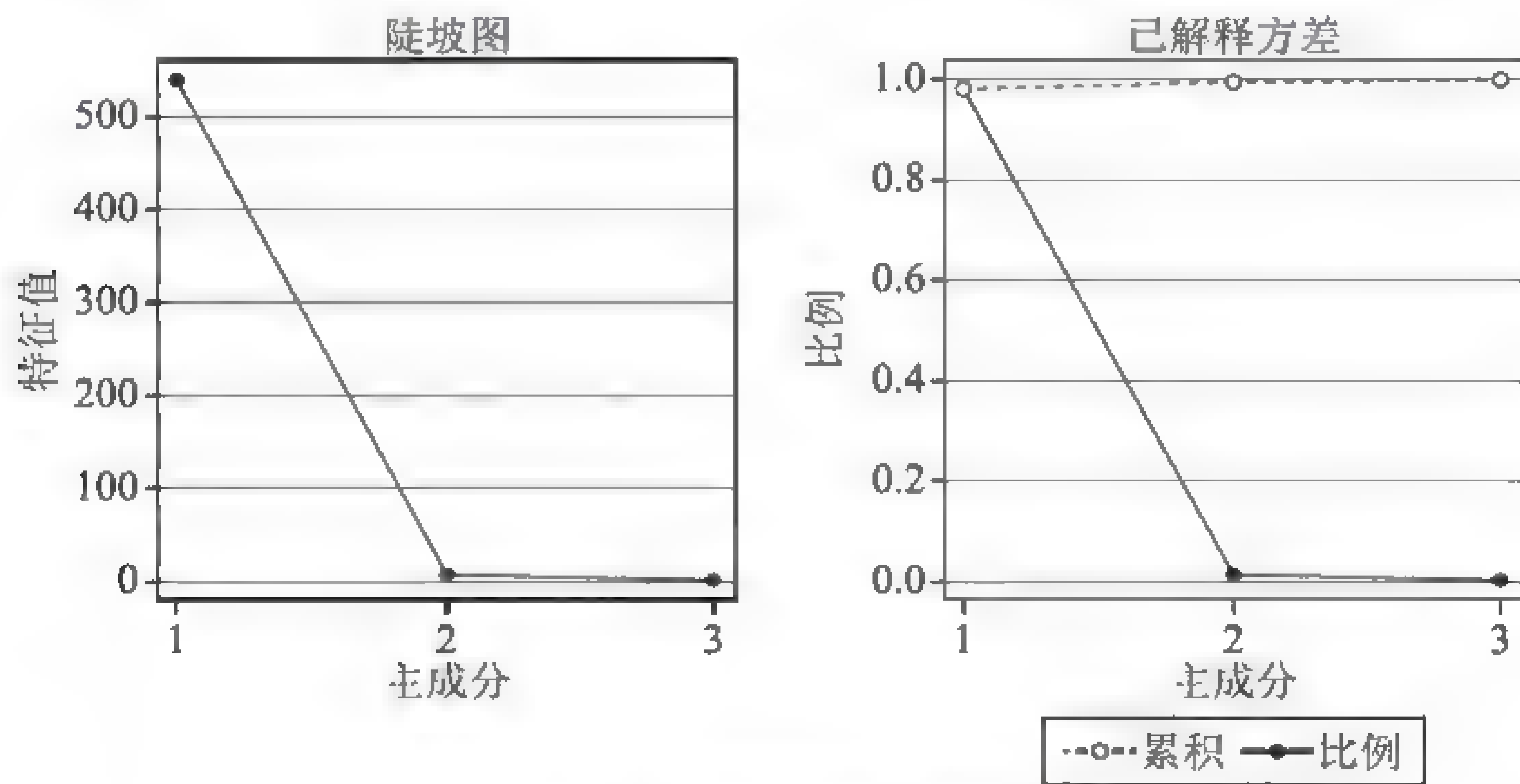


图 23-7　陡坡图和方差图

上面为采用协方差矩阵进行主成分分析，可以看出变量 WEIGHT 在主成分 PRIN1 中的系数最大。原因是变量 WEIGHT 的数值量级比较大（其平均值为 100 左右），而变量 AGE 数值较小（其平均值为 13）。为了消除量纲和单位的影响，我们通常需要将数据标准化（变换后数据的均值为 0，方差为 1）后再执行主成分分析，才不会出现这种偏差。程序 23-4 首先标准化 3 个量化原变量 AGE、HEIGHT、WEIGHT，然后再调用协方差方法进行主成分分析：

程序23-4　数据标准化后用协方差矩阵做主成分分析

```
proc standard data = sashelp.class out class_std mean = 0 std 1;
  var Age Height Weight;
run;

proc princomp data=class_std cov ;
run;
```

由数学上推导可知，基于标准化数据计算的协方差矩阵等价于相关系数矩阵，因此上面的程序输出完全等价于如下使用相关系数矩阵进行的主成分分析，这也是SAS默认的主成分分析方法选项（见程序23-5）。

程序23-5　默认使用相关系数矩阵做主成分分析

```
proc princomp data=sashelp.class ;
run;
```

从相关系数矩阵（见图23-8）可以看出，HEIGHT和WEIGHT关系最紧密为0.8778，其次为HEIGHT和AGE 0.8114，再其次为WEIGHT和AGE 0.7409。

从特征值累积方差贡献上也可以看出只需要一个主成分PRIN1即可达到87.38%(0.8738)，此时主成分与原变量的标准化数据有如下关系：

```
Prin1 = 0.560811 * Age_Std + 0.593307 * Height_Std + 0.577476 * Weight_Std
```

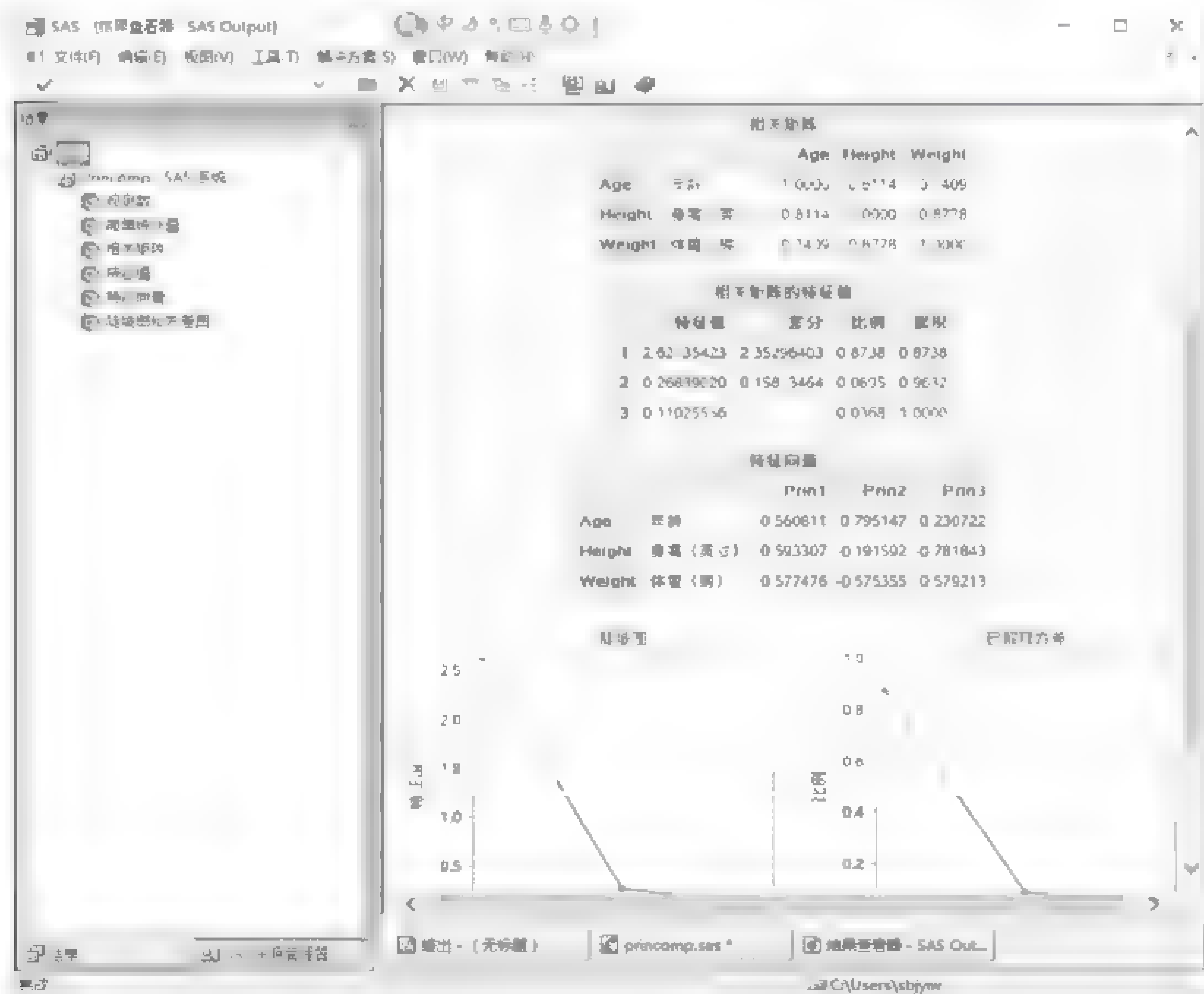


图23-8　基于相关系数矩阵的主成分分析

主成分分析的统计量和结果可以输出到数据集中（见程序23-6），OUTSTAT 将各种统计量包括相关系数，特征值和特征向量（即各主成分和对应变量的系数）输出到CLASS_OUTSTAT中，OUT 将原变量和对应的主成分输出到指定数据集CLASS_OUT中。

程序23-6　输出统计量主成分得分

```
proc princomp data sashelp.class outstat class outstat out=class out;
run;
proc print data=class outstat;
run;
```

上面的过程步输出如下各种统计量（见图 23-9）。

Obs	_TYPE_	_NAME_	Age	Height	Weight
1	MEAN		13.3158	62.3368	100.026
2	STD		1.4927	5.1271	22.774
3	N		19.0000	19.0000	19.000
4	CORR	Age	1.0000	0.8114	0.741
5	CORR	Height	0.8114	1.0000	0.878
6	CORR	Weight	0.7409	0.8778	1.000
7	EIGENVAL		2.6214	0.2684	0.110
8	SCORE	Prin1	0.5608	0.5933	0.577
9	SCORE	Prin2	0.7951	-0.1916	-0.575
10	SCORE	Prin3	0.2307	-0.7818	0.579

图 23-9　输出统计量

输出的数据集 CLASS_OUT 中包括 3 个原始变量和 3 个新的主成分得分，用如下方法验证各主成分之间是互相独立的，即各主成分是线性无关的，其单元格中第一排的相关系数为 0（见程序 23-7 和图 23-10）。

程序23-7　验证主成分之间的相关性

```
proc corr data=class_out out=class_out_corr_prin_prin;
  var prin1-prin3;
  with prin1-prin3;
run ;
```

Pearson 相关系数, N = 19
Prob > |r| under H0: Rho=0

	Prin1	Prin2	Prin3
Prin1	1.00000	0.00000 1.0000	0.00000 1.0000
Prin2	0.00000 1.0000	1.00000	0.00000 1.0000
Prin3	0.00000 1.0000	0.00000 1.0000	1.00000

图 23-10　主成分之间的相关系数

同理，可以检查各主成分和原来的各变量之间的相关关系（见程序 23-8），输出结果（见图 23-11）的第 1 行 (0.90799) 表示因子载荷，反映了主成分受某个原始变量影响的程度，绝对值越大表示影响越强烈，而符号则表示受影响的方向，正相关还是负相关。第 2 行 (<0.0001) 表示零假设下的检验概率，表示在 0.01% 水平上显著。

程序23-8　验证主成分和变量之间的相关关系

```
proc corr data class out out class out corr prin var;
  var prin1 prin3;
  with Age Height Weight;
run ;
```

Pearson 相关系数, N = 19
Prob > |r| under H0: Rho=0

	Prin1	Prin2	Prin3
Age 年龄	0.90799 <0.0001	0.41194 0.0797	0.07661 0.7552
Height 身高（英寸）	0.96060 <0.0001	-0.09926 0.6860	-0.25961 0.2831
Weight 体重（磅）	0.93497 <0.0001	-0.29807 0.2152	0.19233 0.4302

图 23-11　主成分和变量之间的相关关系

最后，可使用程序 23-9 所示的方法将主成分分析的结果 PRIN1 和 NAME，SEX 列输出至 CLASS_PRIN 中，这就是最终降维后的数据集（见图 23-12），可供进一步的回归或聚类等分析处理。但需要记住的一点是，原来的 3 个变量 AGE、WEIGHT、HEIGHT 的信息只有 87.38% 被保留到降维后的数据集中。如果我们需要更多的信息，比如 96.32%，可以选择保留 2 个主成分 PRIN1、PRIN2 到结果数据集中。

程序23-9　输出主成分数据

```
data class_prin;
  set class_out;
  keep name sex prin1;
run;
proc print data=class_prin;
run;
```

Obs	Name	Sex	Prin1
1	阿尔弗雷德	男	1.34442
2	爱丽丝	女	-1.20046
3	芭芭拉	女	0.17287
4	凯露	女	0.37339
5	亨利	男	0.45439
6	詹姆斯	男	-1.50895
7	简	女	-1.18162
8	雅妮特	女	0.96795
9	杰弗瑞	男	-0.50614
10	约翰	男	-0.89384
11	乔伊斯	女	-3.40308
12	莱迪	女	0.23001
13	罗伊斯	女	-1.77682
14	玛丽	女	1.41815
15	菲利普	男	3.39388
16	罗伯特	男	0.50001
17	罗纳德	男	2.00851
18	托马斯	男	-1.81080
19	威廉	男	1.41815

图 23-12　输出主成分数据

23.2　因子分析

23.2.1　因子分析原理

因子分析（也称公因子分析，Common Factor Analysis）的核心思想是寻找观测到的变量背后那些控制数据变异特征、不可观测或未能观测到的隐性变量。在因子分析中这些观测到的变量称为表征变量（Manifest Variable），而不可观测的隐性变量（Latern Variable）则称为因子（Factor），包括各变量共享的公共因子以及各变量特定的特殊因子。因子分析注重分析变量数据的内部结构特征，其目的是试图以少数几个不可观测的隐性变量，来分解和解释变量之间存在的相关性或协方差。因子分析通常需要使用旋转变换，即对公因子应用非奇异性的线性变换来帮助对因子的解释。因子分析的数学模型可概括为

$$X-\mu=AF+\varepsilon$$

$$\text{即}X_i=f(F_1,F_2,\cdots,F_m)\equiv X_i=\mu_i+\sum_{j=1}^{m}a_{ij}F_j+\varepsilon_i \text{ 其中}(i=1,2,\cdots,p,m<p)$$

与主成分分析试图构建变量的线性组合不同，因子分析是试图找到控制观测变量背后的共同因子。因此因子分析的最终结果——因子得分不是变量的线性组合，而是隐性变量的估计量。上面的数学模型中 F_j 是不可观测的变量，称为公因子；系数矩阵 α_{ij} 称为因子载荷，表示因子能预测表征变量的权重；而 ε_i 称为特殊因子，是表示公因子之外变量 X_i 所特有的随机作用部分。因此因子分析是用隐性公因子和随机影响变量的线性组合来表示原变量，是原变量的分解；而主成分分析结果的主成分是原变量的线性组合，是原变量的综合归纳。但两种分析方法都是从协方差矩阵或相关系数矩阵出发的降维分析方法。

23.2.2　巴特利球度检验和 KMO 检验

要进行因子分析，其前提条件是各变量不能相互独立，如果所有变量都相互独立是无法进行因子分析的。因此，首先需要检查各变量是否存在相关性。检查相关性可以用巴特利球度检验（Bartlett Test of Sphericity）和 KMO 检验（Kaiser-Meyer-Olkin Measure of Sampling Adequacy）。

巴特利球度检验的零假设 H_0 为相关系数矩阵与单位矩阵没有显著差异，即相关系数矩阵是单位矩阵，各变量没有公因子。备择假设 H_1 为相关系数与单位矩阵存在显著差异，相关系数不是单位矩阵，各变量至少存在一个公因子。如果巴特利球度检验的统计量较大且对应的概率值小于给定的显著性水平，或 P-值小于 0.05，应该拒绝零假设，表示可以做因子分析；否则不能拒绝零假设，表示各变量之间相互独立，没有公因子，

不适合进行因子分析。

在 SAS 环境中，如下程序 23-10 对 PROC FACTOR 过程步使用 METHOD=ML 和 HEYWOOD 选项即可进行巴特利球度检验（见图 23-13）。

程序23-10　巴特利球度检验
```
proc factor data=sashelp.class method=ml heywood;
  title "Bartlett test of sphericity";
run;
```

系统输出信息的"显著性检验"信息如下，检查 *P*- 值小于 0.0001 表示需要拒绝零假设"H_0: 无公因子"，则应该接受备择假设"HA: 至少一个公因子"，表示可以做因子分析。

基于 19 观测的显著性检验

检验	自由度	卡方	Pr > 卡方
H0: 无公因子	3	41.3313	<0.0001
HA: 至少一个公因子			
H0: 1 个因子足够	0	0.0000	<0.0001
HA: 需要更多因子			

图 23-13　巴特利球度检验

KMO 检验全称为 KMO 抽样适度测定值，是检测 Kaiser 抽样适当性的统计量。它通过比较所有变量间的简单相关系数和偏相关系数的大小来判断变量间的相关性，即

KMO= 相关系数平方和 /（相关系数平方和 + 偏相关系数平方和）

当所有变量间的简单相关系数平方和远大于偏相关系数平方和时，由上面的公式可知 KMO 值接近于 1，表示变量间的相关性越强，原变量集越适合做因子分析；否则，当所有变量间的简单相关系数平方接近于 0 时，KMO 值也接近于 0，变量间的相关性越弱，原变量集不适合做因子分析。通常情况下，因子分析要求 KMO 值一般大于0.7，0.7~0.8 表示适合，0.8~0.9 表示很适合，0.9~1.0 表示非常适合；0.6~0.7 表示不太适合进行因子分析，而 0.5 以下则表示极不适合进行因子分析。在 SAS 中，通过 PROC FACTOR 上使用 MSA 选项，即可输出抽样适当性检测结果（见程序 23-11）。

程序23-11　KMO 检验
```
proc factor data=sashelp.class msa;
  title "KMO Test";
run;
```

系统输出的"抽样适当"检测结果如下（见图 23-14）。这里输出 MSA 0.72301602 表示 SASHELP.CLASS 适合进行因子分析，但也没达到"很适合"或"非常适合"做因子分析的程度。

Kaiser 抽样适当性测度：总体 MSA = 0.72301602

Age	Height	Weight
0.82216940	0.65662258	0.72240279

图 23-14　KMO 检验结果

完成因子分析的前提条件检测后，才能进入因子分析的下一环节，即确定因子个数。一般的准则是要求因子分析的特征值大于 1，且方差累计贡献率在 100% 以上进行综合判断。因子的可解释性准则包括：①每个因子至少有 3 个因子负载，即 1 个因子至少向 3 个表征变量贡献方差；②来自同一因子的变量应该具有共同的概念含义；③根据经验认为归属不同因子的变量衡量了不同的内部结构；④旋转后的因子能够呈现简单的结构特征。

23.2.3　因子分析的具体步骤

在 SAS 中 PROC FACTOR 默认执行的是主成分分析，若要执行因子分析则要求用户指定 PRIORS= 选项或用 PRIORS 语句设置小于 1 的初始共同度。比如下面的例子设置 PRIORS=SMC 来进行因子分析，表示使用 SMC 进行先验公因子方差估计（见程序 23-12）。

程序23-12　执行因子分析

```
proc factor data=sashelp.class priors=SMC;
  title "Factor analysis";
run;
```

它采用先验公因子方差估计，各变量的方差贡献如下（见图 23-15），其总和正是缩减相关矩阵的特征值总计 2.26293856（见图 23-16）。

先验公因子方差估计: SMC		
Age	Height	Weight
0.66199499	0.82803863	0.77290494

图 23-15　变量的 SMC 方差估计

缩减相关矩阵的特征值: 总计 = 2.26293856 平均值 = 0.75431285				
	特征值	差分	比例	累积
1	2.38071107	2.40617119	1.0520	1.0520
2	-0.02546012	0.06685227	-0.0113	1.0408
3	-0.09231239		-0.0408	1.0000

图 23-16　相关矩阵的特征值

系统输出的“特征值”信息如图 23-16 所示。从累积方差贡献比例一列可以看出，只需要保留 1 个因子就累计了 1.0520 的方差贡献。也就是说，只要保留 1 个因子 FACTOR1，其方差贡献量为 2.38071107，因子 2 和因子 3 基本没有什么贡献，而且还是负值。因子分析的特征值跟主成分分析的特征值不同，它可出现负值，因此其阈值也通常是要求累计在 1.0 即 100% 以上，而不是主成分分析方法中定义的 0.8（即 80%）。

因子分析的目的不仅是找到公因子和对变量的分组，还在于探索公因子的现实意义并进行解释。在主成分分析中，每个主成分对应的系数是唯一的。而因子分析中每个因子的系数，即因子载荷 a_{ij} 并不是唯一的。当有两个公因子以上时，初始载荷矩阵可以左乘一个正交矩阵来进行所谓的因子旋转，目的是让旋转后的因子载荷尽可能接近 0 或 1，就是让矩阵元素的平方值向 0 和 1 进行两极分化，但旋转后的公因子之间仍然保持独立性。常用的旋转方法包括简化对因子解释的方差最大旋转法 VARIMAX，简化对变量解

释的 4 次最大正交旋转 QUARTIMAX 以及等量正交旋转 EAUAMAX 方法。在 SAS 中，可以通过对 PROC FACTOR 指定 ROTATE = VARIMAX | QUARTIMAX | EQUAMAX 来指定旋转方法，SAS 支持二十余种不同的因子旋转方法。程序 23-13 展示了因子分析中的指定方差最大旋转方法，由于 SASHELP.CLASS 只有 1 个公因子无法展示旋转，此处改用演示数据 SOCIOECONOMICS 进行说明。

程序23-13 方差最大旋转进行因子分析

```
data SocioEconomics;
  input Population School Employment Services HouseValue;
  datalines;
5700     12.8      2500      270       25000
1000     10.9       600       10       10000
3400      8.8      1000       10        9000
3800     13.6      1700      140       25000
4000     12.8      1600      140       25000
8200      8.3      2600       60       12000
1200     11.4       400       10       16000
9100     11.5      3300       60       14000
9900     12.5      3400      180       18000
9600     13.7      3600      390       25000
9600      9.6      3300       80       12000
9400     11.4      4000      100       13000
;

ods graphics on;
proc factor data=SocioEconomics
  priors=smc msa residual
  rotate=varimax /*方差最大旋转，以简化对因子的解释*/
  outstat=fact_all
  plots=(scree initloadings preloadings loadings);
run;
ods graphics off;
```

图 23-17 为初始因子模式和旋转因子模式，通过旋转因子可看出 POPULATION 和 EMPLOYMENT 能够更好地被因子 1 进行解释，而 HOUSEVALUE 和 SCHOOL 更好地被因子 2 解释。

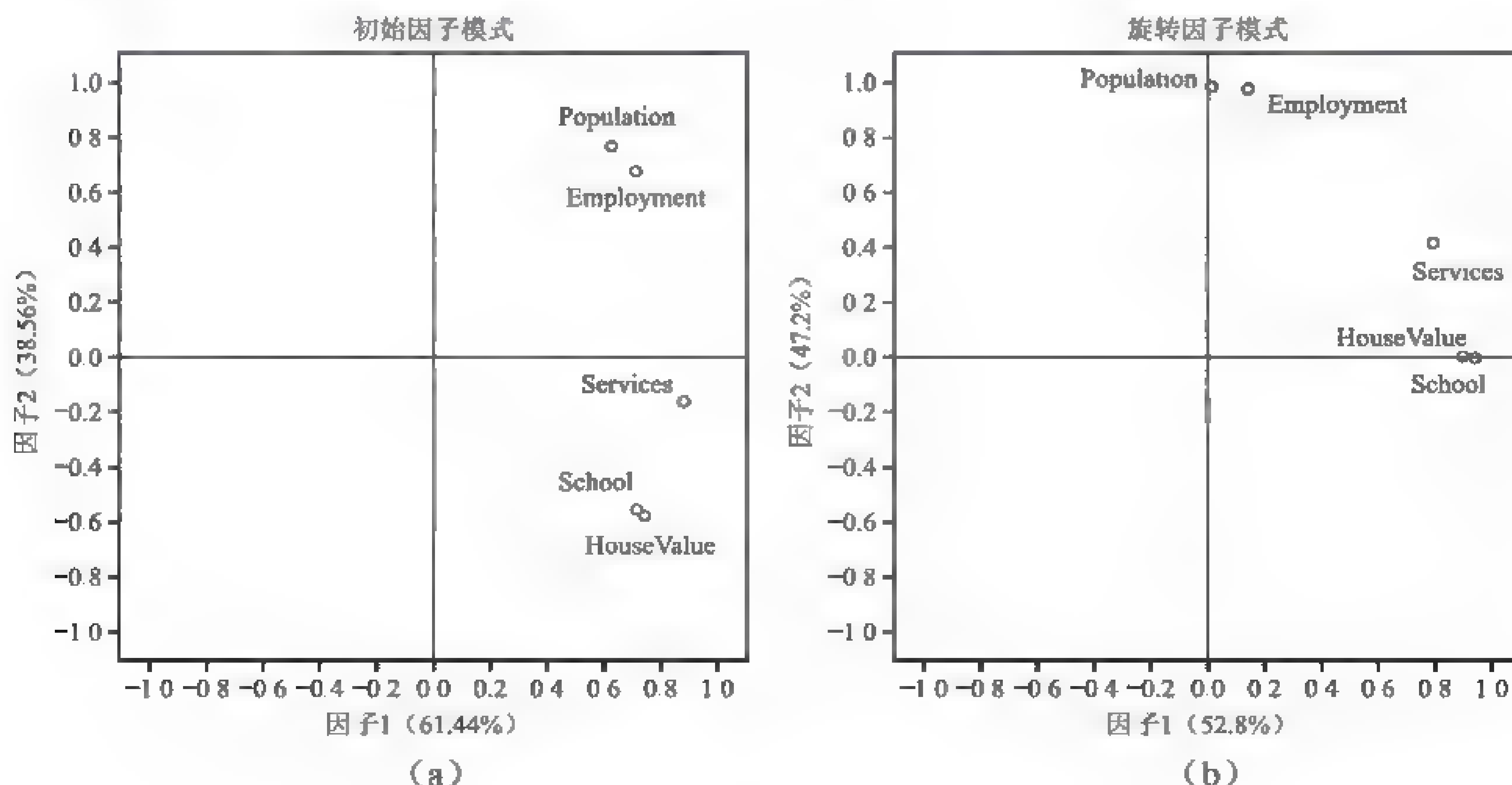


图 23-17 初始因子模式和旋转因子模式

图 23-18 旋转因子模式中我们可看出变量和因子之间的关系。从图中可以看出 SCHOOL 和 HOUSEVALUE 受 FACTOR1 支配比较紧密，分别为 0.90419 和 0.94072；而 POPULATION 和 EMPLOYMENT 受 FACTOR 支配较为紧密，分别为 0.98874 和 0.97499。

旋转因子模式		
	Factor1	Factor2
Population	0.02255	0.98874
School	0.90419	0.00055
Employment	0.14625	0.97499
Services	0.79085	0.41509
HouseValue	0.94072	-0.00004

图 23-18　旋转因子模式

根据图 23-18 的输出信息，它揭示了观测到的表征变量和隐藏因子之间具有如下线性关系：

```
Population = 0.02255 * Factor1 + 0.98874 * Factor2;
School     = 0.90419 * Factor1 + 0.00055 * Factor2;
Employment = 0.14625 * Factor1 + 0.97499 * Factor2;
Services   = 0.79085 * Factor1 + 0.41509 * Factor2;
HouseValue = 0.94072 * Factor1 - 0.00004 * Factor2;
```

因子分析的最终结果是为了计算因子得分，而因子得分可当作自变量进行进一步的回归和聚类等分析处理。在 SAS 中，在 PROC FACTOR 上使用 OUT= 选项和 NFACTORS= 选项来指定结果数据集的名称和保留的因子数目（见程序 23-14）。

程序23-14　输出因子得分

```
ods graphics on;
proc factor data=SocioEconomics
  priors=smc msa residual
  rotate=varimax
  outstat=fact_all
  out=fact_out nfactors=2/*输出因子得分*/
  plots=(scree initloadings preloadings loadings);
run;
ods graphics off;

proc print data=fact_out; run;
```

在系统输出结果（见图 23-19）的“标准化评分系数”信息中，它包含了因子得分的系数关系。

标准化评分系数		
	Factor1	Factor2
Population	-0.33457	0.57427
School	0.28299	-0.05835
Employment	0.25118	0.43751
Services	0.22543	-0.02388
HouseValue	0.49804	0.00268

图 23-19　标准化评分系数

图 23-19 标准化评分系数揭示了因子得分和原变量之间的关系：

```
Factor1 = -0.33457 * Population_Std + 0.28299 * School_Std + 0.25118 *
  Employment_Std + 0.22543 * Services_Std + 0.49804 * HouseValue_Std

Factor2 =  0.57427 * Population_Std - 0.05835 * School_Std + 0.43751 *
  Employment_Std - 0.02388 * Services_Std + 0.00268 * HouseValue_Std
```

其中每一个自变量为原始变量的标准化结果，如 POPULATION_STD 跟 POPULATION 有如下关系：

```
Population_Std=(Population - Population的均值) / Population的标准差
```

当我们打印因子分析输出结果进行验证时，因子得分（FACTOR1 和 FACTOR2 列）和原变量之间关系似乎不正确（见图 23-20）。根本原因是默认打印出来的原变量并没有标准化，而 SAS 因子分析的因子得分是根据标准化数据计算出来的。

Obs	Population	School	Employment	Services	HouseValue	Factor1	Factor2
1	5700	12.8	2500	270	25000	1.21989	-0.10367
2	1000	10.9	600	10	10000	-0.69168	-1.44824
3	3400	8.8	1000	10	9000	-1.25502	-0.83842
4	3800	13.6	1700	140	25000	1.11451	-0.70197
5	4000	12.8	1600	140	25000	0.94810	-0.67770
6	8200	8.3	2600	60	12000	-1.14455	0.53407
7	1200	11.4	400	10	16000	-0.20311	-1.49916
8	9100	11.5	3300	60	14000	-0.42711	0.82738
9	9900	12.5	3400	180	18000	0.22197	0.94027
10	9600	13.7	3600	390	25000	1.44113	0.88080
11	9600	9.6	3300	80	12000	-0.89390	0.96791
12	9400	11.4	4000	100	13000	-0.33023	1.11874

图 23-20　原始变量与因子得分

为此，首先将 SOCIOECONOMICS 数据集标准化为均值为 0，方差为 1 的数据，并输出到 SOCIOECONOMICS_STD 数据集中，然后再使用 SOCIOECONOMICS_STD 进行因子分析（见程序 23-15）并输出结果，此时会发现因子得分和原变量之间的关系就跟标准化评分系数表中列出的线性关系完全匹配（见图 23-21）。这说明 SAS 因子分析内部是使用标准化数据进行处理的，它严谨地消除了变量量纲和数量级的影响。

程序23-15　数据标准化后做因子分析

```
proc standard data=SocioEconomics out=SocioEconomics_std mean=0 std=1;
run;

ods graphics on;
proc factor data=SocioEconomics_std
  priors=smc msa residual
  rotate=varimax
  outstat=fact_all
  out=fact_out nfactors=2
  plots=(scree initloadings preloadings loadings);
run;
ods graphics off;

proc print data=fact_out;run;
```

Obs	Population	School	Employment	Services	HouseValue	Factor1	Factor2
1	-0.15746	0.76031	0.13428	1.29792	1.25637	1.21989	-0.10367
2	-1.52374	-0.30319	-1.39649	-0.96438	-1.09933	-0.69168	-1.44824
3	-0.82607	-1.47865	-1.07422	-0.96438	-1.25637	-1.25502	-0.83842
4	-0.70979	1.20810	-0.51025	0.16677	1.25637	1.11451	-0.70197
5	-0.65165	0.76031	-0.59082	0.16677	1.25637	0.94810	-0.67770
6	0.56928	-1.75852	0.21484	-0.52932	-0.78523	-1.14455	0.53407
7	-1.46560	-0.02332	-1.55762	-0.96438	-0.15705	-0.20311	-1.49916
8	0.83091	0.03265	0.77881	-0.52932	-0.47114	-0.42711	0.82738
9	1.06347	0.59239	0.85938	0.51482	0.15705	0.22197	0.94027
10	0.97626	1.26408	1.02051	2.34206	1.25637	1.44113	0.88080
11	0.97626	-1.03085	0.77881	-0.35530	-0.78523	-0.89390	0.96791
12	0.91812	0.02332	1.34277	-0.18127	0.62819	0.33023	1.11874

图 23-21　标准化数据与因子得分

对于最后输出的因子得分数据，可以完全可以用它代替原来的 5 个变量作进一步的数据分析。

第24章

相关分析与回归分析

24.1 变量关系

变量之间的关系包括确定性的函数关系与非确定性经验关系两种，前者是变量之间能用精确数学模型进行描述的确定性函数关系，如产品销售金额等于产品销售数量 C 乘以销售单价 P，任何一个圆的面积等于半径 R 的平方乘以 π 值。后者则是大量实际观测或试验中建立起来的一种经验性统计关系，称为相关关系（Correlation）。在现实中，确定性函数关系和不确定性的相关关系间并不存在绝对的鸿沟，因为存在确定性函数关系的变量间，也有可能由于随机测量误差或其他因素的干扰而表现为相关关系，而对存在相关关系的变量长期观测具有深刻理解后，这种不确定的相关关系也可借助某个函数关系来进行描述。

相关关系本身说明变量之间存在关联（Association），但它们之间具体如何关联需要进一步分析，它包括平行关系和依存关系两种。如果观测的变量处于平等地位，它们都是随机变量，首先要分析的是两个变量之间是否存在线性相关以及它们相关的程度，采用的分析手段为相关分析（Correlation Analysis）。

如果变量之间已经确定存在显著的相关关系，但其中某个变量的变化，在时间上依存或受控于另外某个或某些变量的变化，则称该变量为因变量（也叫响应变量，常记为 Y），而那些独立变化并导致因变量变化的那些变量被称为自变量（也叫独立变量，常记为 X_i）。自变量 X_i 可以解释并预测因变量 Y 的变化，这时需要采取的数据分析手段为回归分析（Regression Analysis）。回归分析目的是利用观测到的数据试图建立变量之间的经验公式，以实现用自变量来解释并预测因变量的目的，说明自变量和因变量之间存在依存关系。

举个具体的例子，清晨日出与公鸡打鸣这两种现象每天都发生，而且公鸡打鸣可能发生在凌晨 4~5 点钟，时间上可能与日出同时，甚至更早一点。人们注意到这两种现象存在某种关联，但这两种现象之间存在的是平行的关联关系，并非因为公鸡打鸣才导致太阳公公起床来照耀大地。如我们去健身俱乐部运动，运动所消耗的氧气跟运动量是存在依存关系的，运动时间越长，运动量越大则一定会导致身体所消耗的氧气量增大，此时，耗氧量的大小是依赖于运动量的变化而存在的，因此它们之间就存在依存关系。

相关分析和回归分析存在紧密的联系，相关分析是回归分析的基础和前提，回归分

析是相关分析的深入和继续。只有进行相关分析后证明了变量之间存在显著的相关程度，才有进一步做回归分析的必要，否则也没有什么意义。回归分析试图探明变量之间的相关具体是何种表现形式，各个自变量是如何控制因变量的观测值。回归分析的结果会建立一个回归模型，利用该模型可以对因变量进行预测和评估。

相关关系中如果一个变量的变化伴随着另一个变量大致均匀的变动，我们称两者是线性相关；相反，如果一个变量伴随着另一变量的变化产生不均匀的变动，此时两者关系的投影图往往表现为各种曲线形式，则称变量之间为非线性相关。根据研究的变量数目，如果仅涉及两个变量之间的相关，称为简单相关；如果某个变量的变化与两个以上的其他变量的变化相关，称为复相关；当某个变量的变化与若干其他变量的变化相关时，为了研究该变量与其中某个自变量的相关性，我们需要剔除其他变量的影响再研究它们之间的相关性，这种相关分析称为偏相关分析。

24.2　相关分析

24.2.1　线性相关性度量

为了衡量变量之间的相关性，统计学家们引入多种衡量相关性的度量，不同的度量可用于不同测量尺度下的观测变量。

如果两个连续变量服从正态分布，此时基于数据可以计算出协方差和标准差，此时衡量变量之间线性相关性最常用的度量为皮尔逊积矩相关系数（Pearson Product Moment Correlation，简称皮尔逊相关系数，国内也称皮尔逊积差相关系数）。基于样本计算出来的相关系数用 r 表示，而总体的相关系数则用 ρ 表示。其计算公式如下：

$$r=\frac{\delta_{xy}^{\ 2}}{\delta_x\delta_y}=\frac{\sum_i(x_i-\overline{x})(y_i-\overline{y})}{\sqrt{\sum_i(x_i-\overline{x})^2}\sqrt{\sum(y_i-\overline{y})^2}}$$

式中：$\delta_{xy}^{\ 2}$ 为变量 x 和 y 的协方差；δ_x 和 δ_y 分别为变量 x 和 y 的标准差。对于标准化数据，由于其均值为 0，方差为 1（即分母为 1），此时相关系数等价于协方差系数。皮尔逊相关系数也可由如下公式计算：

$$r=\frac{n\sum x_iy_i-\sum x_iy_i}{\sqrt{n\sum x_i^{\ 2}-(\sum x_i)^2}\sqrt{n\sum y_i^{\ 2}-(\sum y_i)^2}}$$

皮尔逊相关系数 r 取值在 -1 到 1 之间，其绝对值 $|r|$ 越靠近 1 表示相关性越强，越靠近 0 表示相关性越弱；符号为正表示两者是正相关，符号为负表示负相关。如果 r=1 表示完全正相关，如果 r=-1 表示完全负相关，r=0 则表示两者没有相关性。如果把变量观测值看作是一个向量，即 X={x_1, x_2, …, x_n}，Y={y_1, y_2, …, y_n} 则皮尔逊相关系数等价于向量 X 和 Y 中心化后的向量夹角余弦值，即中心化向量的余弦相似度 Cos < X, Y > =

$X \cdot Y/(|X| \cdot |Y|)$ 等于两个向量的内积除以两个向量模的乘积。

当我们得到皮尔逊相关系数 r 后，不能因为它的绝对值靠近 1 就认为两个变量之间存在很强的线性相关关系，而应做必要的检验。皮尔逊相关系数的概率值服从自由度为 n-2 的 t 分布

$$t = \sqrt{n-2}\frac{r}{\sqrt{1-r^2}} \sim t(n-2)$$

因此，我们可以对线性相关系数 r 根据其概率值作 t 检验，另外一种方法也可以直接查“积差相关系数临界值表”。

（1）对线性相关系数 r 作 t 检验的零假设 H_0 为 $\rho=0$，即变量之间不相关。而备择假设 H_1 为 $\rho \neq 0$，即变量之间相关。对于上面的统计量 t，由于它服从给定显著性水平 α 下自由度为 n-2 的 t- 分布。因此如果根据样本计算出来的统计量 $|r| > t_{\alpha/2}(n-2)$，我们就有理由拒绝零假设 H_0，认为变量之间存在线性相关性；否则应接受零假设 H_0。需要特别注意的一点是这里应该采用双侧检验，因此使用 $t_{\alpha/2}(n-2)$。

（2）如果直接查积差相关系数临界值表，只要根据样本数据计算出来的线性相关系数 r 的绝对值 $|r|$ 大于临界值表中给定显著性水平 α=0.05 和自由度 df=n-2 下积差相关系数临界值，即可认为变量之间的相关性是显著的。如果 $|r|$ 大于临界值表中显著性水平 α=0.01 和自由度 df=n-2 的临界值，则认为其相关性是非常显著的。

另外还需要注意，考察变量之间的线性相关不仅仅要看 $|r|$ 值本身，还应该考虑样本容量 n 的大小。原因是当样本容量较小时计算出来的 $|r|$ 值容易偏大，而样本容量较大时算出的 $|r|$ 又容易偏小（这一点从临界值表的分布中也可以直接看出），因此仅凭线性相关系数 $|r|$ 来判断变量之间线性相关的密切程度是不恰当的。我们需要结合样本容量 n 进行考虑，避免因为样本量 n 较小没有足够判定性而错误地以为变量之间存在密切的线性相关性。

线性相关系数 r 的平方 r^2 称为判定系数 R 方，虽然由于平方运算后变为正数，不再能衡量相关性的方向（正相关 / 负相关），但它取值在 $0 \leqslant r^2 \leqslant 1$ 之间，往往在回归分析中用来衡量变量总离差的平方和。判定系数 R 方的取值越大表明回归模型拟合程度越好，在理论上判定系数 R 方反映了回归模型中因变量的变化有多少是由模型中的自变量所引起的，从而可以衡量回归模型的有效性。

24.2.2 非参数关联度量

对于分类型数据，由于它是类别数据不能直接计算协方差和标准差，此时也就无法直接计算皮尔逊相关系数。根据非参数统计的思想，我们需要把类别数据根据等级进行排序，也就是说用它们的秩 (Rank) 来代表类别变量值进行相关分析，这种以秩为基础计算的相关系数称为等级相关系数或秩相关系数。等级相关系数以定序变量之间的顺序而不是其数值的大小为基础，利用皮尔逊相关系数计算公式即可求得等级相关系数。

常用的非参数关联度量包括斯皮尔曼秩相关系数、肯德尔 Tau-b 系数以及霍夫丁 D

统计量。

（1）斯皮尔曼秩相关系数（Spearman Rank-Order Correlation）用于衡量两个定序变量之间的相关程度。斯皮尔曼秩相关系数是分布无关的相关性度量，它跟皮尔逊相关系数要求数据服从正态分布的前提不同，它不需要这一前提，因此具有更加广泛的使用领域。

$$r_s = \frac{\sum_i (R_{xi} - \bar{R}_x)(R_{yi} - \bar{R}_y)}{\sqrt{\sum_i (R_{xi} - \bar{R}_x)^2} \sqrt{\sum_i (R_{yi} - \bar{R}_y)^2}}$$

式中：R_{xi} 和 R_{yi} 分别表示变量 x 和 y 第 i 观测值所对应的秩。斯皮尔曼秩相关系数的概率跟皮尔逊相关系数一样服从对应 t- 分布：

$$t = \sqrt{n-2}\frac{r}{\sqrt{1-r^2}} \sim t_{\alpha/2}(n-2)$$

因此也可以对秩相关系数 r_s 作同样的 t- 检验，也可以查“秩相关系数检验的临界值表”进行检验。比如给定显著性水平 α=0.05，如果计算得到的斯皮尔曼秩相关系数大于等于查表所得的数值，则认为有 95% 的置信度认为两个变量相关；如果 α=0.01 则有 99% 的置信度认为两个变量显著相关。有的教材也使用如下公式计算斯皮尔曼秩相关系数。

$$r_s = 1 - \frac{6\sum_i (R_{xi} - R_{yi})^2}{n(n^2-1)}$$

式中：$R_{xi}-R_{yi}$ 表示两个定序变量对应的秩次差。

（2）肯德尔 Tau-b 系数（Kendall tau-b Coefficients）又称肯德尔和谐系数，是基于配对观测变化的一致和非一致的次数，以衡量变量关联的非参数度量。当配对观测中变量变化方向（增大或减小）一致时称为一致性，否则配对观测变化方向不同（其中一个增大另一个减小）时称为不一致性。它常用于计算多个定序变量之间的相关程度，其计算公式如下：

$$\tau = \frac{C-D}{\sqrt{(T_0-T_1)(T_0-T_2)}} = \frac{\sum_{i<j}(\text{sgn}(x_i - x_j)\text{sgn}(y_i - y_j))}{\sqrt{(T_0-T_1)(T_0-T_2)}}$$

公式中的分子 $C-D$ 实际是计算拥有一致性的元素对的数量 C 减去拥有不一致性的元素对的数量 D，其简捷计算方法为统计符号乘积的计数，公式中 sgn(z) 取符号函数取值为（-1，0，1）$T_0=n(n-1)/2$，$T_1=\sum_{k=1}^{s} t_k(t_k-1)/2$ 和 $T_2=\sum_{l=1}^{I} u_l(u_l-1)/2$。$T_0$ 与样本容量有关，而 T_1、T_2 分别与变量 x 和 y 有关。比如计算 T_1，它首先将所有 x 中所有相同元素归类形成 s 个子集（没有相同元素则不形成子集），t_k 表示的是第 k 个子集中所包含的相同元素的个数。T_2 与 T_1 类似但基于变量 y 进行计算。

比如变量 x 和 y 都有 n 个观测，则第 i 个观测可用 (x_i, y_i) 表示，当任意两个观测 (x_i, y_i) 和 (x_j, y_j)。如果 $x_i < x_j$ 且 $y_i < y_j$ 或者 $x_i > x_j$ 且 $y_i > y_j$，则认为这两个元素的变化是一致的；否则如果 $x_i < x_j$ 且 $y_i > y_j$ 或者 $x_i > x_j$ 且 $y_i < y_j$，则认为这两个元素的变化是不一致的。若 $x_i = x_j$ 或 $y_i = y_j$，则认为这两个元素既不是一致，也不是不一致。

下面以例子（见图 24-1）说明如何计算肯德尔 Tau-b 系数，对于 5 个观测的样本，

可形成10个观测对，两个变量之间形成了一个上三角和下三角对称的矩阵，矩阵元素反映两个变量的增减变化方向是否一致。

OBS	X	Y	j=1	j=2	j=3	j=4	j=5
i=1	1	5		不一致	不一致	不一致	不一致
i=2	2	2			不一致	一致	一致
i=3	3	1				一致	一致
i=4	4	4					不一致
i=5	5	3					

图 24-1　计算肯德尔 Tau-b 系数

表中有4个一致和6个不一致，而 T_1 和 T_2 为零，代入计算公式可得

$$\tau=\frac{\sum_{i<j}\left[\operatorname{sgn}(x_i-x_j)\operatorname{sgn}(y_i-y_j)\right]}{\sqrt{(T_0-T_1)}\sqrt{(T_0-T_2)}}=\frac{-6+4}{5(5-1)/2}=-0.2$$

当变量 X 和变量 Y 各个元素都是唯一时，即不存在相同值时，肯德尔系数 Tau-b 可以简化如下：

$$\tau=\frac{C-D}{n(n-1)/2}$$

式中：C、D 分别为拥有一致性的元素对的数目和不一致的元素对的数目，且此时前面公式中的 T_1 和 T_2 均为零。

另外，肯德尔 Tau-b 系数也可由如下公式计算出来，其中 $\sum i$ 表示等级换位总次数：

$$r_k=1-\frac{4\sum i}{n(n-1)}$$

如果使用几何图示表示（见图 24-2），等级换位次数 $\sum i$ 实际上就是它们排列连线的交点数目。

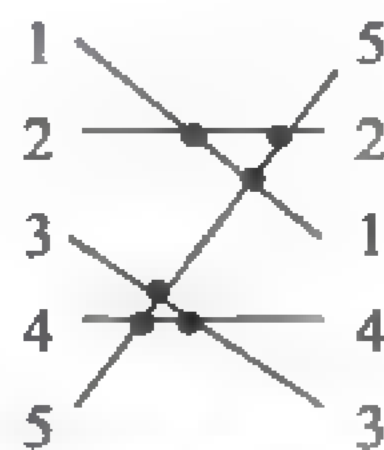


图 24-2　基于换位次数计算肯德尔 Tau-b 系数

$$r_k=1-\frac{4\sum i}{n(n-1)}=1-\frac{4\times 6}{5(5-1)}=-0.2$$

肯德尔 Tau-b 系数的概率计算公式比较复杂，且肯德尔偏 Tau-b 的采样分布不可知。

（3）霍夫丁 D 统计量 (Hoeffding's measure of dependence) 是用于检测对独立性更一般的偏离，它也是衡量变量关联的非参数度量，近似于对 2×2 分类表的卡方统计量观测的加权总和，每组值都是类别的分割点。其计算公式如下：

$$r_h = 30\frac{(n-2)(n-3)D_1 + D_2 - 2(n-2)D_3}{n(n-1)(n-2)(n-3)(n-4)}$$

式中：$D_1=\sum_i(Q_i-1)(Q_i-1)$，$D_2=\sum_i(R_{xi}-1)(R_{xi}-2)(R_{yi}-1)(R_{yi}-2)$，$D_3=\sum_i(R_{xi}-2)(R_{yi}-2)(Q_i-1)$，其中 R_{xi} 和 R_{yi} 分别表示变量 x_i 和 y_i 对应的秩；Q_i 为双变量等级，表示 x 和 y 的值都小于第 i 个点的元素个数加 1。

如果只有一个值（x 或 y）相关且小于第 i 个值，则该观测贡献 1/2 到 Q_i。如果两个变量 x 和 y 都相关，则贡献 1/4 到 Q_i。计算时一般先排序来获得 Q_i，然后再按先后变量对 x 和 y 进行排序计算。霍夫丁 D 统计量基于第一个变量的交换次数来计算。

当数据集的观测之间不存在联系时，D 统计量介于 −0.5 到 1 之间，1 表示完全依赖。当观测之间存在联系时，D 统计量可能会导致较小的数值。也就是说一对变量具有相同值时霍夫丁 D 统计量可能小于 1。当一个小的数据集中存在大量联系时，D 统计量可能小于 −0.5。

霍夫丁 D 统计量的概率值可使用 Blum, Kiefer 和 Rosenblatt (1961) 的渐进分布进行计算（其计算公式如下），而当样本数小于 10 时需要查霍夫丁 D 分布表。

$$\frac{(n-1)\pi^4}{60}D + \frac{\pi^4}{72}$$

24.2.3　定量数据的相关分析

在 SAS 中主要以 PROC CORR 进行相关分析（见程序 24-1），默认使用皮尔逊相关系数进行计算：

程序24-1　基于皮尔逊相关系数的相关分析

```
proc corr data=sashelp.class;
  var age weight height;
run;
```

此时系统输出如下（见图 24-3）。

SAS 系统

CORR 过程

3 变量：Age Weight Height

简单统计量

变量	N	均值	标准差	总和	最小值	最大值	标签
Age	19	13.31579	1.49267	253.00000	11.00000	16.00000	年龄
Weight	19	100.02632	22.77393	1901	50.50000	150.00000	体重（磅）
Height	19	62.33684	5.12708	1184	51.30000	72.00000	身高（英寸）

Pearson 相关系数, N = 19
Prob > |r| under H0: Rho=0

	Age	Weight	Height
Age 年龄	1.00000	0.74089 0.0003	0.81143 <0.0001
Weight 体重（磅）	0.74089 0.0003	1.00000	0.87779 <0.0001
Height 身高（英寸）	0.81143 <0.0001	0.87779 <0.0001	1.00000

图 24-3　皮尔逊相关系数

从图 24-3 的第 3 个表格（皮尔逊相关系数）中可以看出，变量 Age 和变量 Weight、Height 之间的相关系数分别为 0.74089 和 0.811143，变量 Weight 和变量 Height 之间的相关系数为 0.87779。其中 t 统计检验的结果显示其 P- 值分别为 0.003, <0.0001 和 <0.0001，它们都小于 0.01，说明它们之间的相关性是十分显著的，其中 Weight 和 Height 之间的相关性 0.87779 最强。我们可以用自由度 $df=19-2$，$A=0.05$ 直接查“积差相关系数界值表”得到的相关系数临界值为 0.456，而 0.87779 > 0.456 说明两者显著相关；进一步查 $df=19-2$, $A=0.01$，得到临界值 0.575，0.87779 依然大于 0.575 说明两者关系确实十分显著。

如果希望自己写程序进行 t 检验，可以使用下面的程序（见程序 24-2）进行。实际上 SAS 还提供各种统计分布的自动查表功能。

程序24-2 作 t 检验

```
data _null_;
  r=0.87779 ; n=19;

  t= sqrt(n-2)* r /sqrt(1-r*r);
  put "统计量 t=" t;

  df=n-2;
  alpha=0.05;
  t_005=TINV(1-alpha/2, df);
  put "查表 t( " df ")a/2=" t_005 " a=" alpha;

  alpha=0.01;
  t_001=TINV(1-alpha/2, df);
  put "查表 t( " df ")a/2=" t_001 " a=" alpha;

  if t > t_001 then do;
    put "结论拒绝 H0: p=0; H1: p<>0. 即在显著性水平 " alpha "下变量相关性非常显著! ";
  end;
  else do;
    if t > t_005 then do;
      put "结论拒绝 H0: p=0; H1: p<>0. 即在显著性水平 " alpha "下变量相关性显著! ";
    end;
    else do;
      put "结论接受 H0: p=0; H1: p<>0. 即变量之间没有相关关系! ";
    end;
  end;
run;
```

当需要使用其他的相关性度量时，可以指定 PROC CORR 的选项 PEARSON、SPEARMAN、KENDALL 和 HOEFFDING，也可以指定 PLOTS=MATRIX(HISTOGRAM) 来输出投影图和直方图以方便查看各变量之间的关系。其代码如程序 24-3 所示。

程序24-3 输出变量间的相关性度量

```
proc corr data=sashelp.class PEARSON SPEARMAN KENDALL HOEFFDING
  plots=matrix(histogram);
  var age weight height;
run;
```

除了前面已经显示的简单统计量，SAS 系统还会输出如下信息（见图 24-4），分别用于输出斯皮尔曼相关系数、肯德尔 Tau-b 相关系数以及霍夫丁 D 统计量。各种秩相关

系数都指示变量 Weight 和变量 Height 之间存在较强的相关关系。

Spearman 相关系数，N = 19
Prob > |r| under H0: Rho=0

	Age	Weight	Height
Age 年龄	1.00000	0.72536 0.0004	0.76906 0.0001
Weight 体重（磅）	0.72536 0.0004	1.00000	0.85576 <0.0001
Height 身高（英寸）	0.76906 0.0001	0.85576 <0.0001	1.00000

Kendall Tau b 相关系数，N = 19
Prob > |tau| under H0: Tau=0

	Age	Weight	Height
Age 年龄	1.00000	0.61692 0.0006	0.62604 0.0004
Weight 体重（磅）	0.61692 0.0006	1.00000	0.71430 <0.0001
Height 身高（英寸）	0.62604 0.0004	0.71430 <0.0001	1.00000

Hoeffding 依赖系数，N = 19
Prob > D under H0: D=0

	Age	Weight	Height
Age 年龄	0.60218 <0.0001	0.20579 0.0003	0.18856 0.0006
Weight 体重（磅）	0.20579 0.0003	0.92646 <0.0001	0.31609 <0.0001
Height 身高（英寸）	0.18856 0.0006	0.31609 <0.0001	0.96172 <0.0001

图 24-4　3 种相关性度量

系统输出的散点图矩阵（见图 24-5）可以直观地呈现变量间的相关性。其中第 2 排第 3 列的散点图可以看出，它反映 Weight 和 Height 之间确实存在较好的相关性。

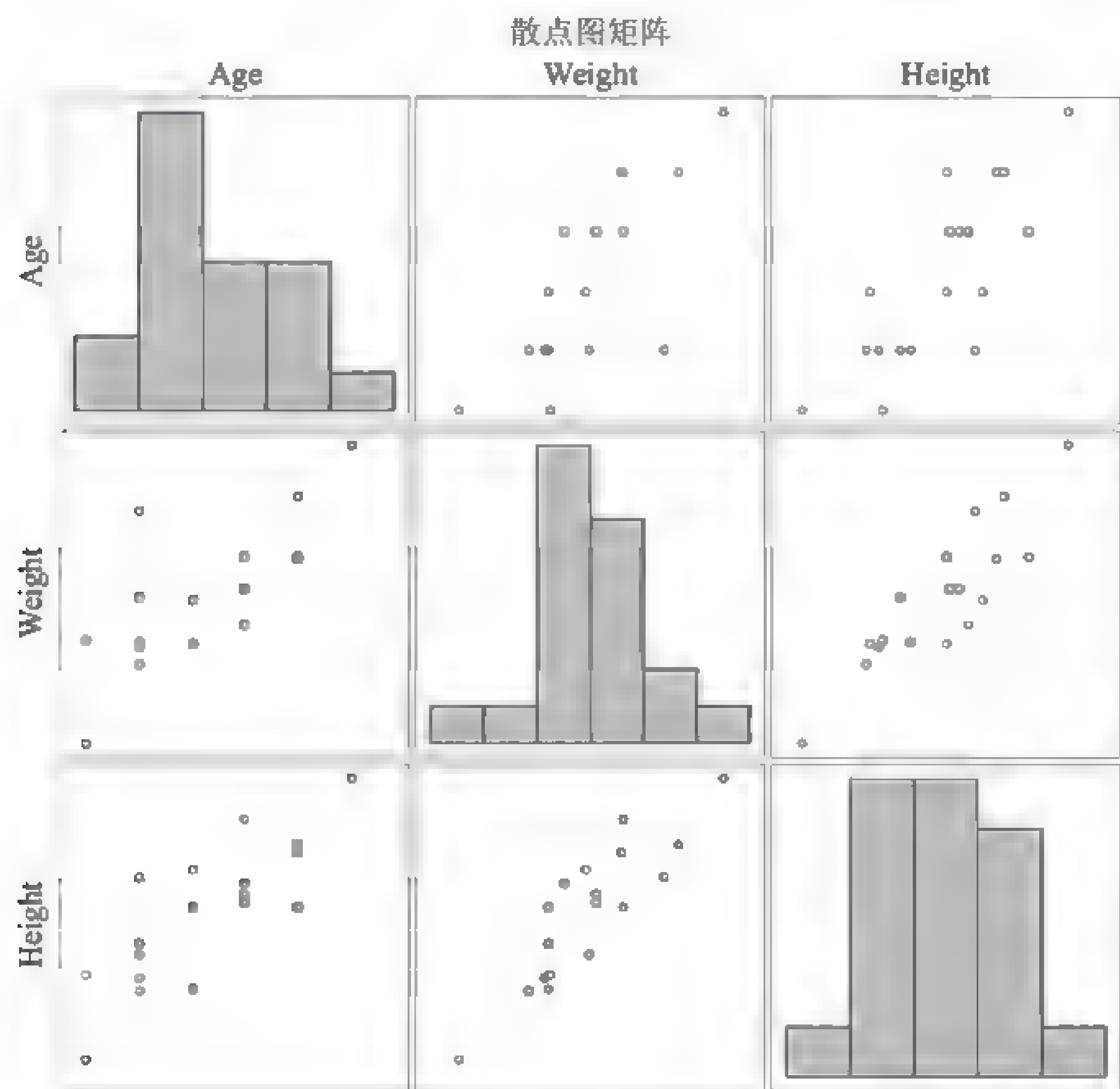


图 24-5　散点图矩阵

24.2.4　类别数据的相关分析

对于类别数据，如所有变量都是文本数据，该如何进行相关分析呢？比如对于下面的例子（见程序 24-4 及其输出图 24-6），该如何分析员工学历和绩效之间的相关性呢？

程序24-4　准备类别数据

```
data mydata;
  input name $ degree $ capacity $;
  label name "姓名" degree "学历" capacity="绩效";
  datalines;
```

```
张三 硕士 良好
李四 本科 良好
王五 本科 优秀
赵六 硕士 一般
钱七 专科 一般
孙八 专科 较差
run;
proc print data=mydata label;
run;
```

Obs	姓名	学历	绩效
1	张三	硕士	良好
2	李四	本科	良好
3	王五	本科	优秀
4	赵六	硕士	一般
5	钱七	专科	一般
6	孙八	专科	较差

图 24-6 类别数据

首先需要把文本数据进行人为排序来获得对应的秩，然后再根据秩做进一步分析。此时我们需要用到自定义输出格式来进行数据转换（见程序 24-5）。

程序24-5 利用自定义格式生成秩数据

```
proc format library=work;
  value  $degree '硕士'=1 '本科'=2 '专科'=3;
  value  $capacity '优秀'=1 '良好'=2 '一般'=3 '较差'=4;
run;

data mydata_rank;/*生成秩数据*/
  set mydata;

  length degree_rank capacity_rank 8;
  label degree_rank="学历排名" capacity_rank="绩效排名";
  degree_rank=put(degree, $degree.);
  capacity_rank=put(capacity, $capacity.);
run;
proc print data=mydata_rank label;
run;
```

得到结果如图 24-7 所示。

Obs	学历排名	绩效排名	姓名	学历	绩效
1	1	2	张三	硕士	良好
2	2	2	李四	本科	良好
3	2	1	王五	本科	优秀
4	1	3	赵六	硕士	一般
5	3	3	钱七	专科	一般
6	3	4	孙八	专科	较差

图 24-7 生成自定义秩数据

这样我们就可以利用斯皮尔曼系数进行分析（见程序 24-6），运行该代码的结果如下（见图 24-8）。

程序24-6　Spearman 相关系数

```
proc corr data mydata rank SPEARMAN;
  var degree rank capacity rank;
run;
```

Spearman 相关系数, N = 6
Prob > |r| under H0: Rho=0

	degree_rank	capacity_rank
degree_rank 学历排名	1.00000	0.43082 0.3938
capacity_rank 绩效排名	0.43082 0.3938	1.00000

图 24-8　Spearman 相关系数

由于类别变量 degree 和 capacity 之间的斯皮尔曼相关系数为 0.43082，且其 *P*- 值 0.3938 大于 0.05，因此需要接受原假设，即两个变量之间的相关性不显著。

如果待分析的变量是量化变量，而不是类别变量，但如果只关注其排名而不是数值本身，则可以使用斯皮尔曼 Spearman 相关系数进行分析，也可以调用 PROC RANK 计算各观测在变量内的排名，然后再使用标准的皮尔逊相关系数进行相关分析。实际上，下面的代码运行结果是等价的（见程序 24-7）。

程序24-7　Spearman 相关系数

```
proc corr data=sashelp.class spearman;
  var age weight  height ;
run;
```

等价于如下程序 24-8：

程序24-8　生成秩然后计算皮尔逊相关系数

```
proc rank data=sashelp.class out=class_rank;
  var age weight height ;
  ranks age_rank weight_rank height_rank;
run;
proc corr data=class_rank  ;
  var age_rank weight_rank height_rank;
run;
```

对于肯德尔 Tau-b 系数和霍夫丁 D 统计量，使用方法都很简单。比如“非参数关联度量”一节中计算肯德尔 Tau-b 系数的例子可用如下 SAS 代码（见程序 24-9）实现，结果显示变量 x,y 之间相的肯德尔 Tau-b 系数为 −0.20000。

程序24-9　肯德尔 Tau-b 系数

```
data mydata;
  input x y @@;
  datalines;
1 5
2 2
3 1
4 4
5 3
run;
proc corr data mydata kendall;
  var x y;
run;
```

24.3 回归分析

如果在相关分析后发现变量之间存在非常显著的相关关系，则可以使用回归分析做进一步的分析处理。下面以 SASHELP.CLASS 为例演示如何进行回归分析。

基于相关分析的内容，我们已经知道 Weight 和 Height 之间存在十分显著的相关性（R=0.87779, P<0.0001）。首先可使用 PROC SGPLOT 或 PROC GPLOT 绘制散点图检查样本数据在两个变量空间上的分布情况，其代码见程序 24-10 和程序 24-11 所示，输出结果如图 24-9 所示。

程序24-10 GPLOT 绘制散点图

```
proc gplot data=sashelp.class;
  title "散点图";
  plot  weight*height ;
run;
```

或

程序24-11 SGPLOT 绘制散点图

```
proc sgplot data=sashelp.class;
  title "散点图";
  scatter x=height y=weight;
run;
```

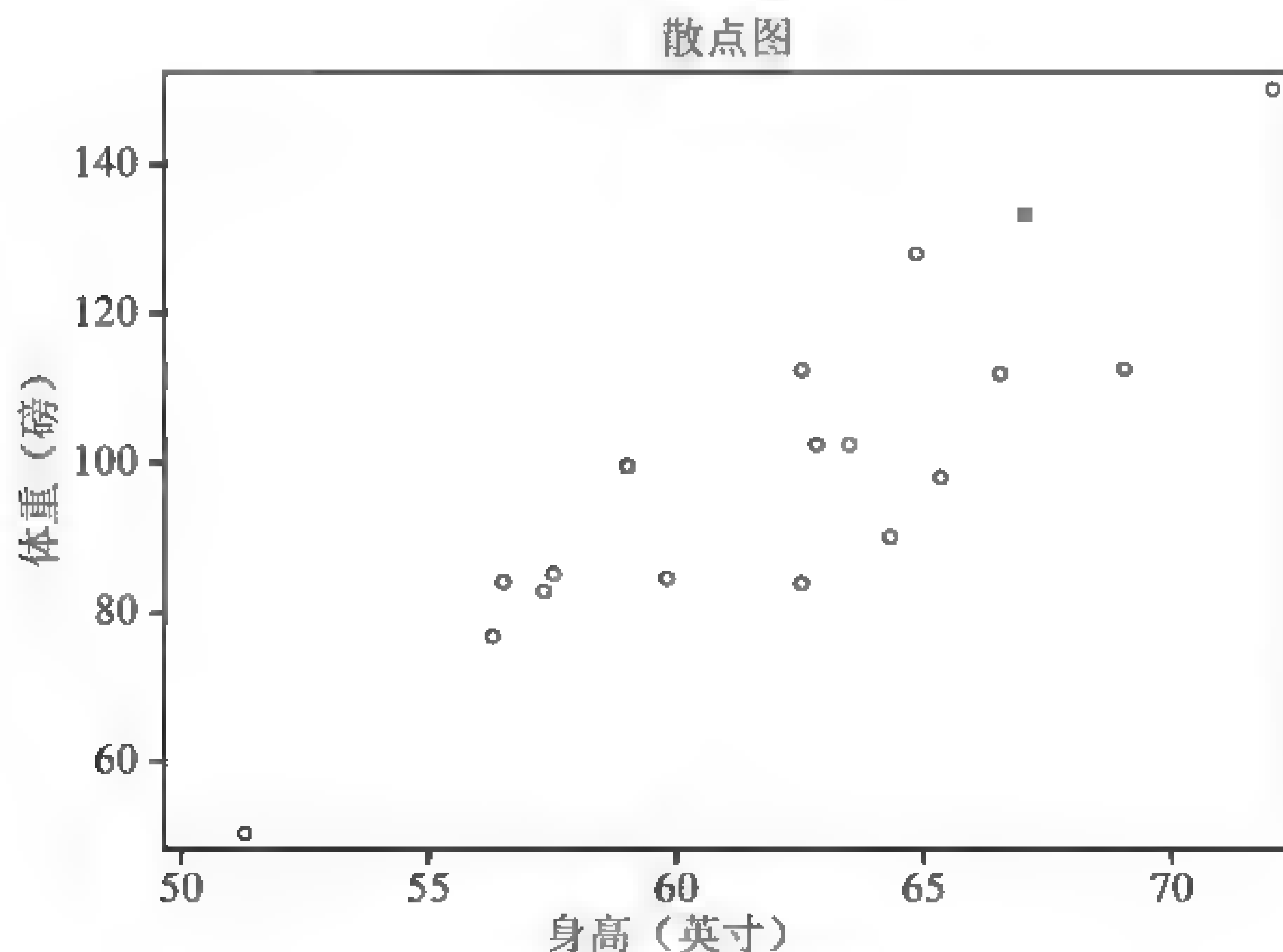


图 24-9 变量空间的散点图

其散点分布特征揭示变量之间存在较强的相关性，但它并不能说明到底是体重依赖于身高，还是身高依赖于体重，变量之间的依存关系是需要根据实际情况进行回归建模。在建模时确立目标响应变量是进行有监督数据分析的第一步。比如，已知班级中学生的身高，是否能很好地预测他们的体重？我们试图建立体重关于身高的函数，可使用 PROC REG 的 MODEL 语句来指定需要拟合的回归模型。目标是建立线性回归方程

Weight $=\hat{\beta}_0+\hat{\beta}_1$*Height，即变量 Weight 关于变量 Height 的函数（见程序 24-12）。

程序24-12　PROC REG 回归分析

```
proc reg data=sashelp.class;
  title "回归分析";
  model Weight=Height;
run;
quit;
```

运行后系统输出如图 24-10 所示。



方差分析

源	自由度	平方和	均方	F 值	Pr > F
模型	1	7193.24912	7193.24912	57.08	<0.0001
误差	17	2142.48772	126.02869		
校正合计	18	9335.73684			

均方根误差	11.22625	R 方	0.7705
因变量均值	100.02632	调整 R 方	0.7570
变异系数	11.22330		

图 24-10　检查方差分析结果

该输出结果的方差分析部分显示，模型自由度为模型参数个数减去 1，简单线性回归模型只有两个参数 $\hat{\beta}_0$ 和 $\hat{\beta}_1$，因此自由度为 1。而校正的总体自由度则等于数据集中的观测数减 1，即 19-1=18。表中还可以看出模型的回归离差平方和 (SSR) 为 7193.24912，而残差平方和 (SSE) 为 2142.48772，它们除以各自的自由度 1 和 17 得到各自的均方差 7193.24912 和 126.02869。有了模型和误差的均方差就可以做 F 检验，其中 F 统计量等于模型均方差 / 误差的均方差，即 F=7193.24912/126.02869=57.08，其对应的 *P*- 值 <0.0001 显示回归模型的 F 统计量非常显著，表明回归模型的解释能力非常显著，模型总体上能对数据变异的绝大部分进行解释。

该输出结果中的均方根误差（Root MSE）值是误差项标准差的估计值，由于误差项总体标准差不可知，要用样本估计量 S_y 代替。它是实际观测值 y 与回归估计值 $\hat{y}$ 离差平方和的均方根，反映实际观测值在回归直线周围的分散状况，也反映回归直线的拟合程度。其计算公式为

$$S_y=\sqrt{\frac{\sum_i^n (y_i-\hat{y}_i)^2}{n-2}}=\sqrt{\frac{\sum_i^n y_i^2-\hat{\beta}_0\sum_i^n y_i-\hat{\beta}_1\sum_i^n x_i y_i}{n-2}}$$

计算得到 $S_y=11.22625$，而表中变异系数则是数据变异在消除量纲影响后可对比的表现形式。

图 24-10 中的 R 方和调整 R 方是两个用来衡量模型拟合程度的统计量，越接近 1 表示拟合越好。拟合优度系数 R 方也称判定系数，它的值表示自变量 Height 解释了

77.05% 的因变量 Weight 的变异。

实际上，数学上已经证明判定系数 R 方等价于变量之间的相关皮尔逊相关系数 r 的平方：从前面相关分析时已经知道变量 Weight 和变量 Height 之间的皮尔逊相关系数为 0.87779，其平方 0.87779×0.87779 刚好等于回归分析拟合优度系数 R 方。它反映回归分析中样本观测值聚集在样本回归直线周围的紧密程度，同时也度量了回归模型能解释回归离差平方和 SSR 在总离差平方和 SST 中所占的比例，即 R 方 $= SSR/SST = 7391.24912/9335.73684 = 0.7705$。然而，实践中 R 方的计算不能代替回归方程总体线性关系的 F 检验，因此 F 检验总是必要的。

分析结果的参数估计表（见图 24-11）显示了回归模型的参数估计结果，第 1 行和第 2 行分别列出了 $\hat{\beta}_0$ 和 $\hat{\beta}_1$ 的参数估计以及对应的标准误差、检验每个参数（$\hat{\beta}_0$ 和 $\hat{\beta}_1$）是否显著的 t 统计量及相应的 P- 值。对回归系数 $\hat{\beta}_0$ 和 $\hat{\beta}_1$ 的显著性 t- 检验结果 P- 值分别为 0.0004 和 < 0.001 表明两个参数都非常显著。

参数估计

变量	标签	自由度	参数估计	标准误差	t 值	Pr > |t|
Intercept	Intercept	1	-143.02692	32.27459	-4.43	0.0004
Height	身高（英寸）	1	3.89903	0.51609	7.55	<0.0001

图 24-11　参数估计

在线性回归分析中，利用已经观测到的样本数据如何计算截距（Intercept）$\hat{\beta}_0$ 和斜率（Slope）$\hat{\beta}_1$ 的数学方法是普通最小二乘法（OrdinaryLeast Square Estimation），其核心思想是寻找截距和斜率参数的估计值，使误差项的平方和（残差）达到最小。比如上面一元线性回归方程的两个参数可由样本数据计算得到，其中 $\bar{x} = \sum_i^n x_i/n, \bar{y} = \sum_i^n y_i/n$ 。

斜率：$\hat{\beta}_1 = (n\sum_i^n x_i y_i - \sum_i^n x_i \sum_i^n y_i)\big/(n\sum_i^n x_i^2 - \sum_i^n x_i) = 3.89903$

截距：$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1\bar{x} = -143.02692$

回归方程的斜率 $\hat{\beta}_1$ 和截距 $\hat{\beta}_0$ 的标准误差 $S_{\hat{\beta}_1}$ 和 $S_{\hat{\beta}_0}$ 都与 S_y 有关，它们标准误差的计算公式如下：

斜率：$S_{\hat{\beta}_1} = S_y\sqrt{1/\sum_{i=1}^n (x_i - \bar{x})^2} = 0.51609$

截距：$S_{\hat{\beta}_0} = S_y\sqrt{1/n + (\bar{x})^2\Big/\sum_{i=1}^n (x_i - \bar{x})^2} = 32.27459$

回归方程斜率和截距在给定显著性水平 α，即 $100\times(1-\alpha)\%$ 置信度下的区间估计上下限分别为 $\hat{\beta}_1 \pm t_{\frac{\alpha}{2}}(n-2)Sy\sqrt{1/\sum_{i=1}^n (x_i - \bar{x})^2}$ 和 $\hat{\beta}_0 \pm t_{\frac{\alpha}{2}}(n-2)S_y\sqrt{1/n + (x)^2\Big/\sum_{i=1}^n (x_i - \bar{x})^2}$ 。

上面的拟合线性模型说明因变量 Weight 和自变量 Height 之间服从如下估计的回归方程：$Y = f(x) = \hat{\beta}_0 + \hat{\beta}_1 \cdot X$，即

$$\text{Weight} = \hat{\beta}_0 + \hat{\beta}_1 \times \text{Height} = -143.02692 + 3.89902 \times \text{Height}$$

比如身高最小的乔伊斯，其身高 Height 和体重 Weight 的观测值为 51.3 和 50.5，将身高代入上面线性回归模型即可估计对应的体重，称为点估计。

$$\text{Weight} = -143.02692 + 3.89903 \times \text{Height} = -143.02692 + 3.89903 \times 51.3 = 56.993319$$

也就是给定身高 Height 51.3 根据回归方程拟合出来的体重 Weight 值为 56.993319。而实际观测到的 Height 值为 50.5，此时该观测值的残差等于实际值减去拟合值 50.5 56.993319=-6.493 319。

回归分析建立的回归方程模型 $Y\ \ f(x)$ 在用于预测与控制时，包括点估计和区间估计两种。

（1）Y 个别值的点估计 - 对于自变量 X 的一个给定值 x_0，可以使用回归方程计算出它的估计值 $\hat{y}_0$。即利用回归方程预测某个给定自变量对应的 Y 值。点估计不能给出估计的精度，因此点估计与实际值之间是有误差的，比如上面的残差 -6.493319 就是点估计的残差值。因为存在误差，所以需要进行区间估计。

（2）Y 平均值的区间估计：也称为置信区间估计，就是对于自变量 X 的一个给定值 x_0 计算因变量 Y 的均值 $E(\hat{y}_0)$ 的估计区间，称为置信区间，置信区间估计的下限和上限分别称为 LCLM 和 UCLM。因变量均值 $E(\hat{y}_0)$ 在 1-α 置信水平下的置信区间为

$$\hat{y}_0 \pm t_{\frac{\alpha}{2}}(n-2)S_y\sqrt{1/n+(x_0-\overline{x})^2 \Big/ \sum_{i=1}^{n}(x_i-\overline{x})^2}$$

式中：S_y 为误差项标准误差的估计。

（3）Y 个别值的区间估计：也称为预测区间估计，就是对于自变量 X 的一个给定值 x_0 计算因变量 Y 的一个个别值的估计区间，称为预测区间。个别值的区间估计不但包含参数估计的变异，也包括误差的变异，因此该区间要比置信区间要宽。预测区间估计的下限和上限分别称为 LCL 和 UCL，因变量个别值在 1-α 置信水平下的预测区间为

$$\hat{y}_0 \pm t_{\frac{\alpha}{2}}(n-2)S_y\sqrt{1+1/n+(x_0-\overline{x})^2 \Big/ \sum_{i=1}^{n}(x_i-\overline{x})^2}$$

在调用 PROC REG 进行回归分析时可同时输出原观测的预测值（即点估计）、残差值（即实际值与预测值的偏差）、置信区间上下限、预测区间上下限等数据。代码如程序 24-13 所示。

程序24-13　输出点估计和区间估计数据

```
proc reg data=sashelp.class;
  title "回归分析";
  model Weight=Height;
  output out=class_out predicted=p residual=r
    LCL=lcl UCL=ucl LCLM=lclm UCLM=uclm ;
run;
proc print data=class_out;
run;
```

运行上面的代码后 SAS 系统会打印输出结果如图 24-12 所示，其中可以看到原来的各观测数据及其点估计和区间估计数据，p 列为基于观测自变量的估计值，r 列为点估计的残差。

Obs	Name	Sex	Age	Height	Weight	p	lclm	uclm	lcl	ucl	r
1	阿尔弗雷德	男	14	69.0	112.5	126.006	116.942	135.071	100.646	151.367	-13.5062
2	爱丽丝	女	13	56.5	84.0	77.268	68.907	85.630	52.150	102.386	6.7317
3	芭芭拉	女	13	65.3	98.0	111.580	105.260	117.899	87.066	136.094	-13.5798
4	凯露	女	14	62.8	102.5	101.832	96.375	107.289	77.526	126.138	0.6678
5	亨利	男	14	63.5	102.5	104.562	98.982	110.141	80.228	128.895	-2.0615
6	詹姆斯	男	12	57.3	83.0	80.388	72.667	88.108	55.476	105.299	2.6125
7	简	女	12	59.8	84.5	90.135	84.040	96.231	65.678	114.592	-5.6351
8	雅妮特	女	15	62.5	112.5	100.662	95.226	106.099	76.361	124.964	11.8375
9	杰弗瑞	男	13	62.5	84.0	100.662	95.226	106.099	76.361	124.964	-16.6625
10	约翰	男	12	59.0	99.5	87.016	80.479	93.552	62.445	111.587	12.4841
11	乔伊斯	女	11	51.3	50.5	56.993	43.804	70.182	29.883	84.103	-6.4933
12	茱迪	女	14	64.3	90.0	107.681	101.842	113.520	83.286	132.075	-17.6807
13	罗伊斯	女	12	56.3	77.0	76.488	67.960	85.017	51.315	101.662	0.5115
14	玛丽	女	15	66.5	112.0	116.259	109.182	123.335	91.539	140.978	-4.2586
15	菲利普	男	16	72.0	150.0	137.703	125.861	149.545	111.223	164.184	12.2967
16	罗伯特	男	12	64.8	128.0	109.630	103.571	115.690	85.182	134.078	18.3698
17	罗纳德	男	15	67.0	133.0	118.208	110.771	125.645	93.383	143.034	14.7919
18	托马斯	男	11	57.5	85.0	81.167	73.600	88.735	56.303	106.032	3.8327
19	威廉	男	15	66.5	112.0	116.259	109.182	123.335	91.539	140.978	-4.2586

图 24-12　输出点估计和区间估计数据

实际上，从过程步 PROC REG 输出的拟合图（见图 24-13）中能清晰反映点估计、置信区间和预测区间的信息。

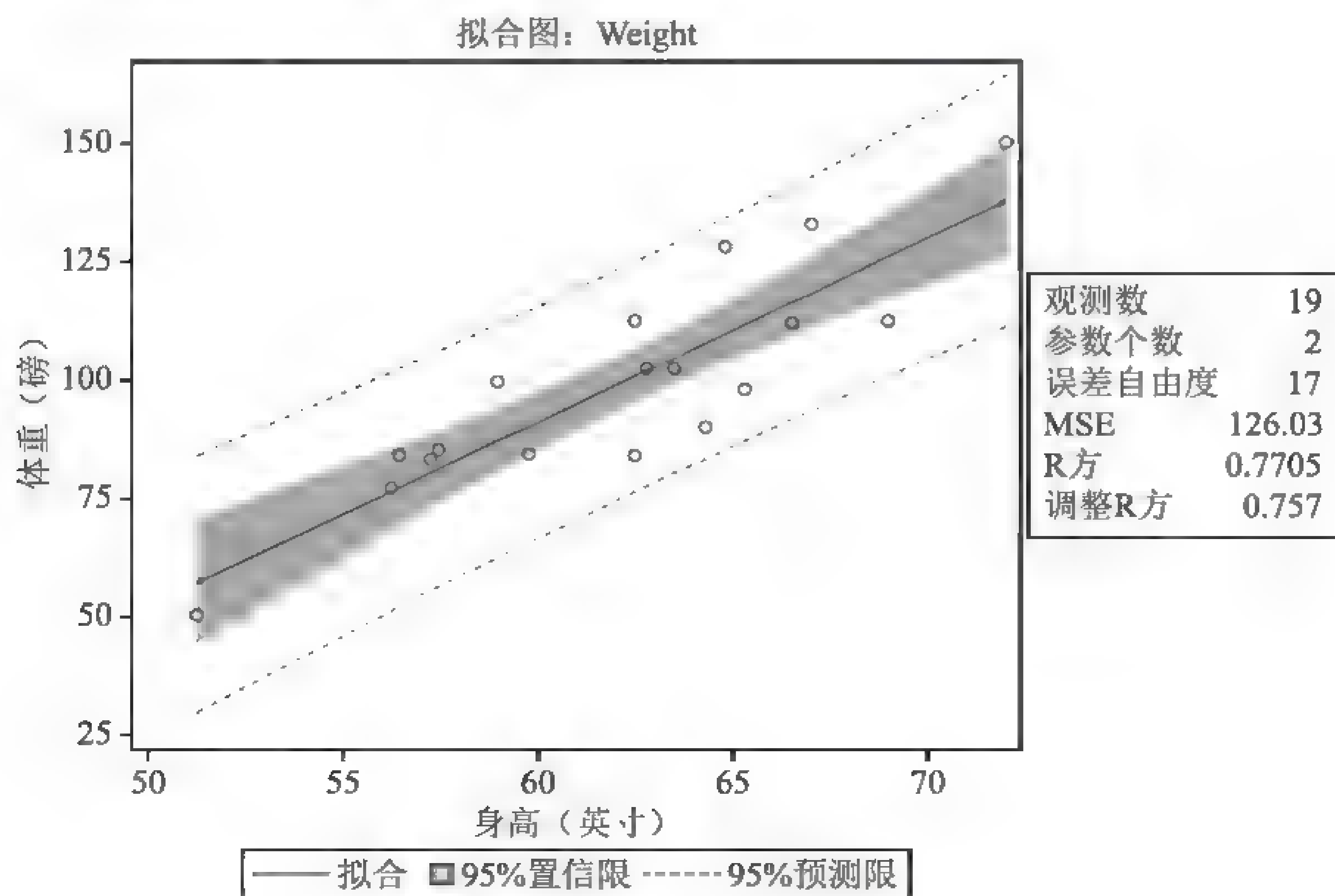


图 24-13　回归分析的拟合图

对 SASHELP.CLASS 数据集进行线性回归模型拟合的结果中调整 R 方等于 0.757，表明体重和身高之间是存在某种线性依存关系的。通过用 Height 对 Weight 进行建模得到的拟合回归模型，对给定身高数据，其中有 95% 的把握认为体重的平均值会落在图中蓝灰色的置信区间内，同样有 95% 的把握认为根据估计的回归方程对个别体重值进行预测，其估计值会落在虚线带内。

回归分析的拟合诊断图部分（见图 24-14）输出非常丰富的信息，如最左上角的残差和预测值投影图可以看到残差的分布情况，它说明数据中变异并不是恒定的，而是随着预测值的增长而略有增长。但总体而言残差没有明显的趋势，因此回归分析总体上是可接受的。但如果残差呈现扇形趋势，说明需要一个方差平稳化的数据变换；而如果呈现曲线趋势（比如半圆形）则暗示回归模型中可能需要一个二次项。实践中回归分析需要根据拟合诊断图来调整优化模型。

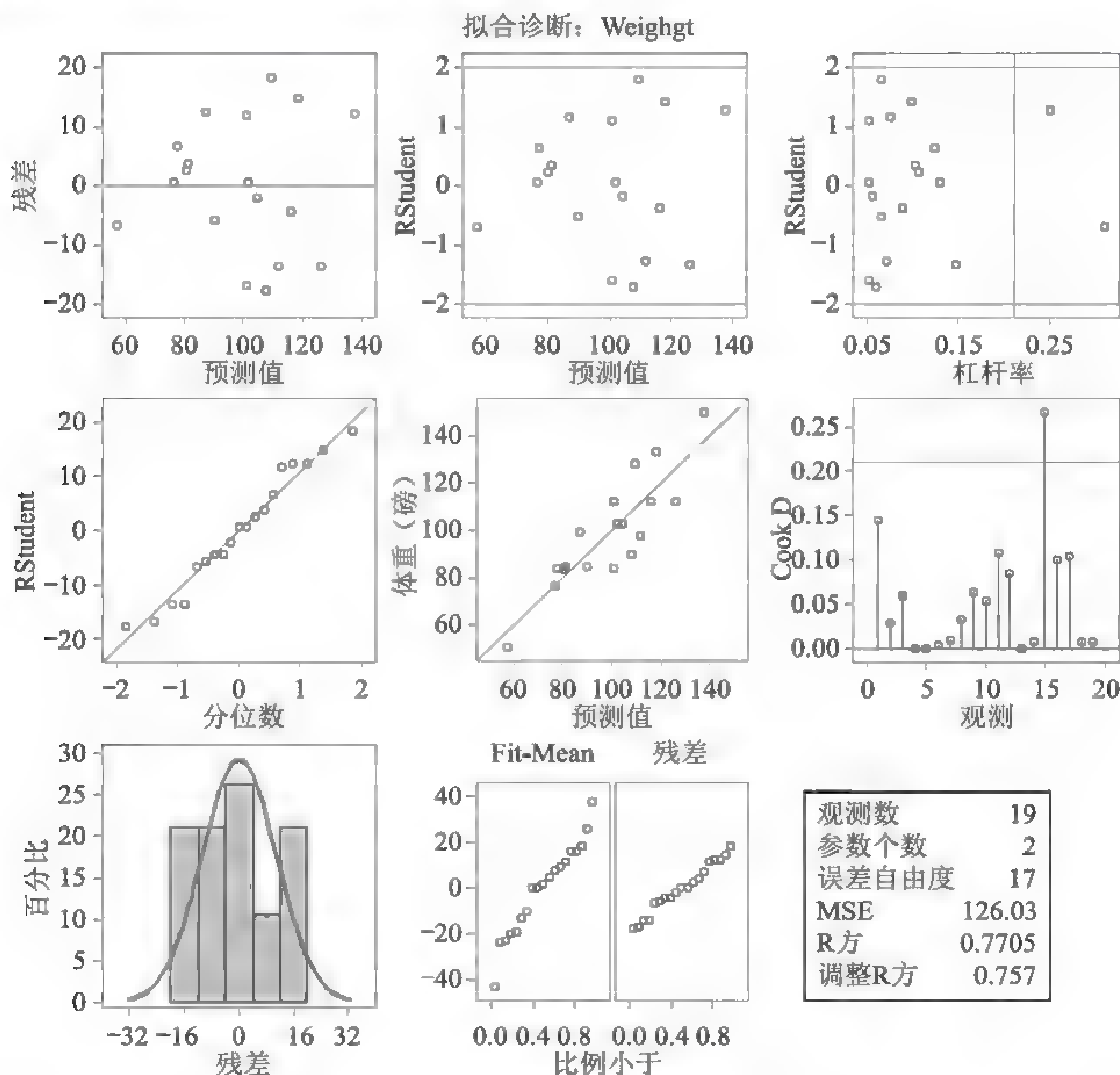


图 24-14 拟合诊断图

回归分析是数据分析科学中极其重要的组成部分，上面介绍的只是最基本的部分。SAS 作为全球最为专业和领先的数据分析系统，除了提供通用的回归分析过程

步 PROC REG 外，它还提供各种广泛用于对单个独立数值变量进行线性回归模型拟合的能力；SAS 专业分析软件包 SAS/STAT 模块中提供各种专业细分的回归分析模型，包括鲁棒回归、广义线性回归、非线性回归、非参数回归、分位数回归、用于调查数据的回归建模、用于生存数据的回归建模以及用于变换变量的回归建模。SAS STAT 模块最常用的是逻辑回归 LOGISTIC 和非线性回归 NLIN 过程步，SAS STAT 总共提供多达 27 种不同的回归模型过程步用于拟合，包括 ADAPTIVEREG、CATMOD、GAM、GENMOD、GLIMMIX、GLM、GLMSELECT、LIFEREG、LOESS、LOGISTIC、MIXED、NLIN、NLMIXED、ORTHOREG、PHREG、PLS、PROBIT、QUANTREG、QUANTSELECT、REG、ROBUSTREG、RSREG、SURVEYLOGISTIC、SURVEYPHREG、SURVEYREG、TPSPLINE 和 TRANSREG；另外，SAS/ETS 模块中则提供各种专门用于计量经济学领域的时间序列分析和联动系统分析的过程步，包括 AUTOREG、PDLREG、MODEL、TSCSREG、SYSLINE 以及 ENTROPY 等，感兴趣的读者可以查看相关帮助来获得各过程步的细节。

程序 24-14 的例子简单展示了如何用 NLIN 非线性回归模型的高斯—牛顿法对数据进行拟合，首先人为用特定函数生成待分析的数据，然后调用 SAS 进行拟合看能不能很好地重建模型并估算模型参数。结果如图 24-15 所示，它则表明 NLIN 能完美拟合输入数据为 $y=a+bx+cx^2$ 函数的样本数据，非线性回归后的参数估计为 $a=4$，$b=3$，$c=2$，与构造样本数据时完全一样。SAS 的非线性回归模型支持近 20 种不同的非线性模型，功能非常强大。

程序24-14　非线性回归

```
data mydata;
  do x=1 to 30;
    y=4 + 3 * x + 2 * x *x;
    output;
  end;
run;

proc nlin data=mydata plots = fit(stats=all)
  plots=diagnostics(stats=all);
  model y=a + b*x + c *x*x;
  parameters a=1 b=1 c=1 ;
run;
```

参数	估计	近似标准误差	近似 95% 置信限	
a	4.0000	2.849246E-13	4.000000	4.000000
b	3.0000	4.2369E-14	3.000000	3.000000
c	2.0000	1.326148E-15	2.000000	2.000000

图 24-15　非线性回归输出的参数估计

第25章

聚类分析

聚类分析用于在事先不知道类别的情况下，完全按照反映对象特征的数据将对象进行分类。它与判别分析不同。判别分析是基于已知分类的样本建立判别函数，对未知类别的个体归属于哪个类别进行判别。因此，聚类分析是一种没有类别信息可参考的情况下，利用距离或者相似性系数将一个集合划分成若干个子集的过程，它是一种无监督的学习过程。

要将对象聚类，首先要定义衡量对象亲疏关系的距离或者相似性度量，也可以是密度阈值。如果对样本进行聚类称为Q型聚类，如果对变量进行聚类称为R型聚类。Q型聚类常用样本之间的距离进行衡量，而变量之间则常用相似性度量。常用的距离包括曼哈顿距离、欧氏距离（欧式距离）、马哈拉诺比斯距离（马氏距离）、切比雪夫距离、闵氏距离（明氏距离）、海明距离等；常见的相似性度量则包括余弦相似度、皮尔逊相关系数、Jaccard相似系数等。

基于划分的方法是选择若干个所谓的质心或凝聚点作为类别的代表，然后将所有待分类对象不断归并到对应类别上的聚类过程。类别的质心可以是真实存在的样本，也可以是样本空间内的虚拟点。经典的*K*-均值及其衍生算法就是一种基于划分的聚类算法，该算法优点是简洁快速，缺点是需要预先人为指定类别数目*k*。*K*-均值的步骤一般是先由用户指定聚类个数*k*，进行随机产生或直接指定*k*个初始聚类中心；然后将每个对象分配给最近的聚类中心，再根据已分配类别的对象重新计算*k*个类别新的聚类中心；重复这一分配/更新步骤直到聚类中心点不再发生变化或迭代中距离改进量小于某个阈值，或者迭代的次数达到指定次数为止。

除了基于划分的方法，形成聚类的过程还可以是将所有对象作为同一类，然后自上而下的逐渐细分，或者将每个对象作为单独分类，然后自下而上不断融合形成同一个类。这种聚类方法称为层次聚类或系统聚类，大多数采用自下而上的方法。它根据距离函数将被分类的对象按照树状结构关联起来，因此我们可以在指定距离上对树状结构进行分割，所有叶子节点就在指定的距离尺度上被分解为若干子集。比如图25-1中的虚线处可以把SASHELP.CLASS的所有学生分成3个簇，乔伊斯自成一簇，罗纳德/罗伯特/菲利普3个观测形成另一簇，其余归入其他簇。

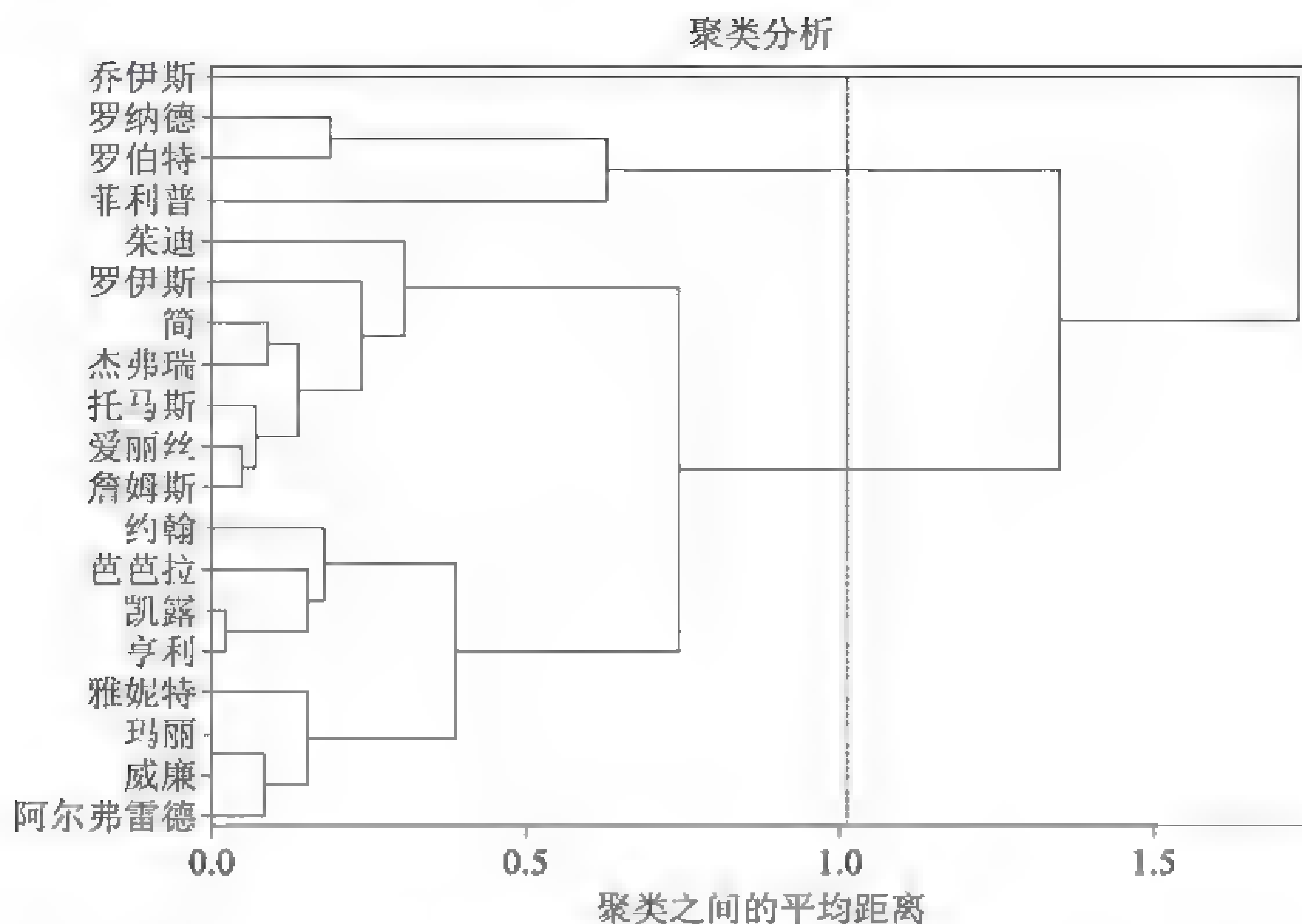


图 25-1 SASHELP.CLASS 聚类图

虽然物以类聚、人以群分的思想已经存在了数千年，但现代聚类分析却始于人类学和心理学。聚类分析并不是一个特定算法，而是一个蕴含分类思想的复杂过程。聚类分析已经发展出上百种聚类方法，但并不存在所谓“正确”的聚类，聚类结果最终还是要通过人为主观经验判断才能恰当应用。到目前为止，尽管某些统计量可对一些不好的聚类结果提供非常有用的信息，但不存在某个理想的统计量可对聚类分析结果的质量进行科学判断。

聚类分析也不存在对各种现实状况通用的所谓万能方法，因此不管是基于划分的方法，如 *K*- 均值及其衍生算法和 CLARA 算法，还是基于层次聚类的方法如 CURE/ROCK/BIRCH 算法都有自己的特定使用场景。比如 *K*- 均值方法就不能处理诸如环形分布的非凸聚类以及基于密度的对象结构，因此人们需要发明基于分布或密度的其他聚类方法。聚类算法的选择和待分析的数据类型，分析目的、计算量以及具体应用有关。

（1）基于分布的聚类方法是跟统计分布最密切的聚类方法，它基于期望最大化思想，其理论基础是最可能来自于同一分布的对象，它们应该归属同一类别。此类方法的计算一般比较复杂且可能过度拟合。

（2）基于密度的聚类方法则认为归属同一类别的数据应该在空间上具有比其他区域更加密集的特性，而稀疏区域的数据倾向于被认为是噪声或离群点。基于密度的典型的聚类方法是 DBSCAN 方法。该方法的缺点是需要基于密度下降来检测聚类边界，而这一点在现实中并不容易保证。

随着大数据时代对数据语义理解需求的增加，而层次聚类方法因计算量大又不适用于大数据，从而催生了一些快速聚类方法，用来对大数据提前生成较粗粒度的数据分区，然后再用较慢的层次聚类方法对各分区数据做进一步的聚类。评估一个现代聚

类算法好坏的标准是，要求能处理高维的大数据，发现任意形状的簇且不需要用户输入太多参数。

25.1 聚类度量

聚类度量是用来度量两个对象（样本或变量）之间的相似性，主要包括距离度量和相似性系数两大类，距离常用于样本的聚类，一般用于定距尺度变量，而相似性系数常用于变量的聚类。其中距离越大表示差别越大，而相似性度量越大则表示差别越小。实践中可以使用 $d_{ij}=1-|r_{ij}|$ 或 $d_{ij}^2=1-r_{ij}^2$ 来将相似性系数 r_{ij} 变换成统一的距离尺度 d_{ij}，距离越大表示变量间差异越大。

假定 X 为 n 行 m 列的数据，对于每一个元素 x_{ij} 有 i=1, 2, …, n，j=1, 2, …, m，其中 n 为样本数，m 为变量数。则常用的距离度量和相似性系数的定义如下：

25.1.1 距离系数

（1）城市街区距离（City Block Distance）：反映在二维平面上就是指上下左右移动时两个点之间的距离，即 $D_{AB}=|x_A-x_B|+|y_A-y_B|$。当你从城市的一个地方沿着街道前进到另一个地方时，不管你是在哪个十字路口转弯，起点和终点之间的距离是不变的。它也称为绝对距离或曼哈顿距离，（见图 25-2 曼哈顿距离），其数学定义如下：

$$D_{ij}=\sum_{k=1}^{m}|x_{ik}-x_{jk}|$$

（2）欧氏距离（Euclidean Distance）：反映在二维平面上就是两点之间的直线距离，即 $D_{AB}=\sqrt{(x_A-x_B)^2+(y_A-y_B)^2}$（见图 25-2 欧几里得距离），反映在三维以上空间中就是两个数据点之间的实际距离。如果把数据点看作向量，欧氏距离也是该维度空间上向量的自然长度。其数学定义如下：

$$D_{ij}=\sqrt{\sum_{k=1}^{m}(x_{ik}-x_{jk})^2}$$

（3）切比雪夫距离（Chebyshev Distance）：反映在二维平面上就是如果一个点只能移到周围 8 个相邻点中的任意一个，则从一个点 A 到另一个点 B 所需的最小步数就是切比雪夫距离，即 $D_{AB}=\max(|x_A-x_B|,|y_A-y_B|)$（见图 25-2 切比雪夫距离）。切比雪夫距离也是两个点在各维度上的差值的最大值，其数学定义如下：

$$D_{ij}=\max(|x_{ik}-x_{jk}|)$$

图 25-2 显示了 3 种距离在二维平面上的具体语义：

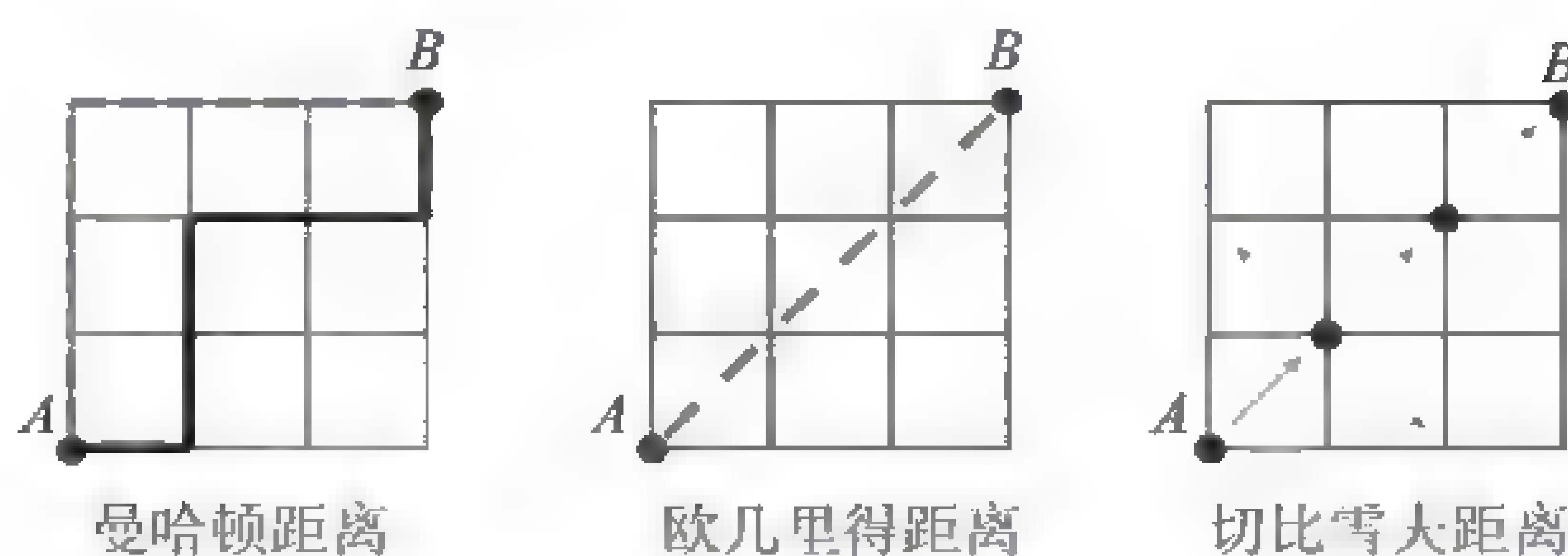


图 25-2　3 种距离的形象表示

（4）闵氏距离（Minkowski Distance）是以上各种形式距离的统一表示，可理解为欧氏距离的一般形式。其数学定义如下：

$$D_{ij}=\left(\sum_{k=1}^{m}\left|x_{ik}-x_{jk}\right|^{p}\right)^{\frac{1}{p}}$$

其中 $p=1$ 即曼哈顿距离，$p=2$ 即欧氏距离，在 $p\to\infty$ 时，就是切比雪夫距离。图 25-3 显示了闵氏距离在不同 p 值时的几何表示（$p=2^h$ $h=-2, -1.5, -1, \cdots$）。可以说切比雪夫距离是闵氏距离的极限形式，而闵氏距离是所有距离的统一表示形式。

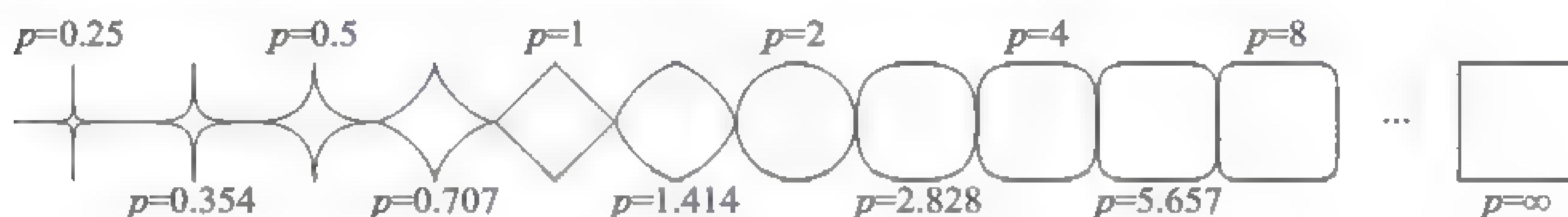


图 25-3　闵氏距离的几何表示

（5）马氏距离（Mahalanobis Distance）：假定两个随机向量 X_i 和 X_j 为来自均值向量为 μ，协方差矩阵为 $\sum$ 的同一分布，则马氏距离可以衡量它们之间的差异程度。其数学定义如下：

$$d_{ij}=\sqrt{(X_i-X_j)'\sum^{-1}(X_i-X_j)}$$

其中如果协方差矩阵为单位矩阵，则马氏距离蜕变为欧氏距离；如果协方差矩阵为对角矩阵（说明各变量之间相互独立），则距离度量称为标准化欧氏距离。

两点之间的马氏距离与原始数据的测量单位无关，且标准化数据和去中心化数据计算出的马氏距离是相同的。马氏距离考虑到各变量之间的联系且尺度无关，因此它能够排除变量之间相关性的干扰且不受量纲的影响。图 25-4 中垂直实线 2.5 到两个分布概率密度曲线相交的垂直虚线之间的观测值 x，尽管看起来离 μ_2 5 更近一点，但实际上它来自于总体 μ_1=0，基于样本分布计算的马氏距离可揭示这一点。

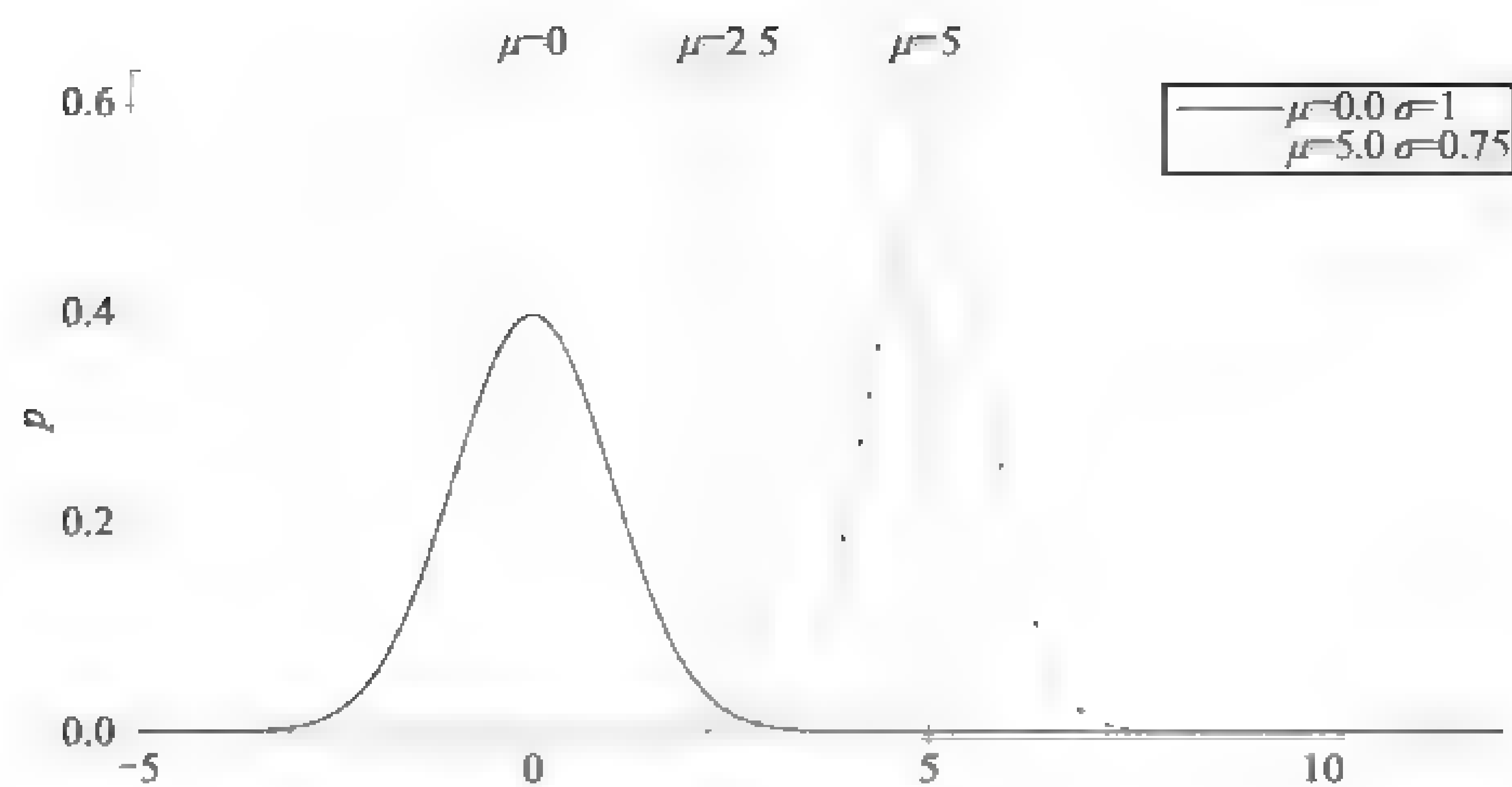


图 25-4　马氏距离

由于马氏距离是基于样本分布的一种距离，如果两个同样的样本来自协方差矩阵不同的两个总体，其计算出来的马氏距离是不同的。马氏距离的物理意义是规范化主成分空间中的欧氏距离，因此也称为广义欧氏距离。马氏距离的缺点是可能放大一些变化微小的变量的作用。

（6）兰氏距离（Canberra Distance）如果观测值都是大于零的数据，即 $X_{ij}>0$，则可根据如下公式计算兰氏距离。兰氏距离自带标准化，能克服数据量纲的影响且对奇异值不敏感，适合高度偏倚的数据处理。其缺点是没有考虑变量之间的相关性。数学定义如下：

$$d_{ij}=\frac{1}{p}\sum_{k=1}^{p}\frac{\left|X_{ik}-X_{jk}\right|}{X_{ik}+X_{jk}}$$

（7）汉明距离（Hamming Distance）用于衡量两个等长字符串之间的距离，就是其中一个字符串变换到另一个字符串所需的最小替换次数，如 “1001” 和 “1111” 之间的汉明距离为 2，而 “1001” 和 “0110” 之间的汉明距离为 4。

25.1.2　相似性/相关系数

上面讲的距离系数用于衡量两个样本或观测的相似程度不同，相似性和相关性系数一般用于衡量纵向上两个变量的相关程度。在第 24 章相关分析中已经讲到了衡量线性相关性的皮尔逊相关系数，该系数在聚类分析中依然具有重要作用。

（1）皮尔逊相关系数：等于两个变量相关程度的协方差除以两个变量的标准差，取值在 [-1, 1]。其计算公式如下：

$$P_{ij}=\frac{\operatorname{cov}(X_i,X_j)}{\sqrt{D(X_i)}\sqrt{D(X_j)}}=\frac{\sum_{k=1}^{n}(x_{ki}-\bar{x}_i)(x_{kj}-\bar{x}_j)}{\sqrt{\sum_{k=1}^{n}(x_{ki}-\bar{x}_i)^2}\sqrt{\sum_{k=1}^{n}(x_{kj}-\bar{x}_j)^2}}\quad 其中\ i,j=1,2,\cdots,m$$

它大于零表示正相关，否则表示负相关；其绝对值越接近 1 表示相关的程度越高，一般按每级 0.2 可分为 5 个等级（见图 25-5）：无相关、弱相关、中等相关、强相关和极强相关。

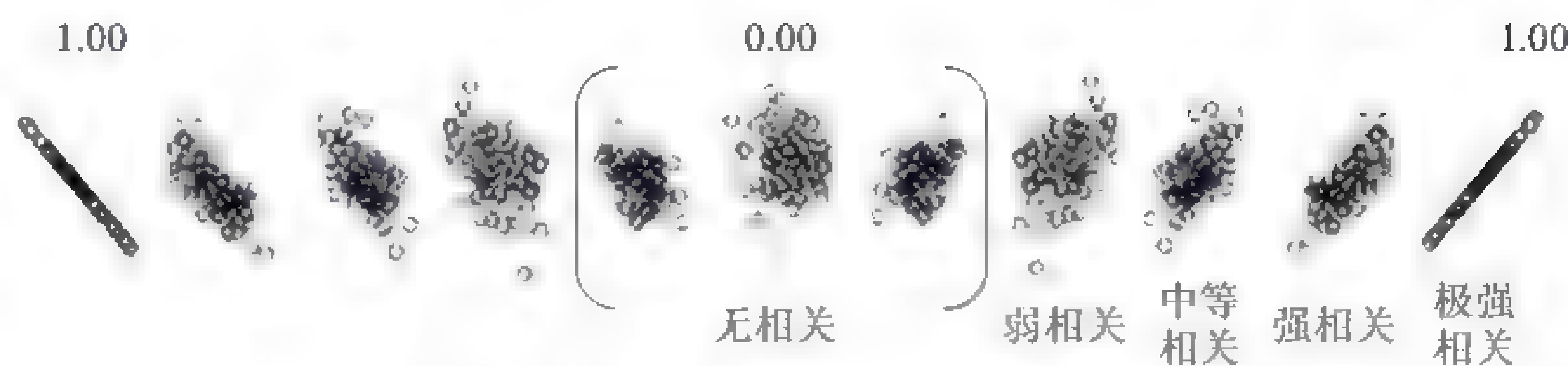


图 25-5 皮尔逊相关等级

皮尔逊相关系数计算公式中每个变量都减去了该变量的均值，这是一个去中心化的处理过程。从公式中可知计算皮尔逊相关系数要求两个变量的标准差（即公式中的分母）都不能为 0。一般要求变量是彼此相互独立的连续型变量，且它们之间存在线性关系，两个变量的总体是正态分布或接近正态分布。

（2）余弦相似度系数：如果把观测数据看作是二维平面上的两个向量，如果只关心向量的方向而不是数值大小，则可用两个向量的夹角余弦 $\mathrm{Cos}(\theta)$ 来衡量两个二维向量的相似性。余弦相似度系数可推广到 m 维向量空间中的两个向量，其取值在 [-1, 1] 之间。余弦相似度系数越大表示两个向量的夹角越小，否则表示两个向量的夹角越大。两个向量方向完全重合时为 1，两个向量的方向完全相反时为 -1。其数学定义如下：

$$S_{ij}=\frac{\sum_{k=1}^{m}x_{ik}x_{jk}}{\sqrt{\sum_{k=1}^{m}{x_{ik}}^{2}}\sqrt{\sum_{k=1}^{m}{x_{jk}}^{2}}}\ ,\ i,j=1,2,\cdots,n$$

如果余弦相似度只关心向量方向而对数值不敏感，实践中往往会导致计算结果误差。比如有两个用户对两个项目的评分分别为（2, 4）和 (8, 10) 分（满分为 10 分），计算余弦相似度为 $\left[(2\times8)+(4\times10)\right]/(\sqrt{2^2+4^2}\sqrt{8^2+10^2})=0.98$。实际上从评分可以看出第一个用户不喜欢那两个项目，而第二个用户很喜欢这两个项目。则说明这儿用余弦相似度存在一定的问题，因此引入修正余弦相似度（Adjusted Consine Similarity），即将计算公式中的每一项都减去其均值来进行归一化，即

$$S_{ij}=\frac{\sum_{k=1}^{m}(x_{ik}-\overline{x}_i)(x_{jk}-\overline{x}_j)}{\sqrt{\sum_{k=1}^{m}(x_{ik}-\overline{x}_i)^2}\sqrt{\sum_{k=1}^{m}(x_{jk}-\overline{x}_j)^2}}$$

两个用户对两个项目的评分均值为 (5, 7)，因此将两个向量修正为 (-3, -3) 和 (3, 3)，此时再计算余弦相似度为 -1，此余弦相似度就是修正余弦相似度，它更能反映两个用户对该两个项目评价的实际情况。

实际上，从公式中可以看出归一化数据计算出来的余弦相似系数等价于皮尔逊相关系数，即 $\mathrm{CosSim}(x-\overline{x},y-\overline{y})=\mathrm{Corr}(x,y)$。而对于均值为 0，方差为 1 的标准化数据，皮尔逊相关系数还等于协方差。

（3）Jaccard 相似系数：两个集合的交集在并集中所占的比例，称为集合的杰卡德相似系数（见图 25-6）。它衡量两个集合的相似程度，可用于衡量样本的相似度。其数

学表示如下，但其中 x_i 和 y_i 是集合而不再是向量。

$$J_{ij}=\frac{x_i\cap x_j}{x_i\cup x_j}$$

如果 p 为两个集合都包含的元素个数，q 为 x_i 中包含而集合 x_j 不包含的元素个数，r 为集合 x_i 中不包含而集合 x_j 包含的元素个数。

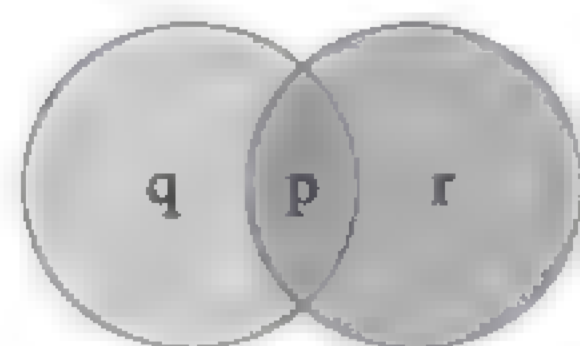


图 25-6　集合关系

则 Jaccard 相似系数可根据如下公式计算出：

$$J_{ij}=\frac{p}{q+p+r}$$

Jaccard 相似系数衡量两个样本中包含相同元素的比例，而 Jaccard 距离则衡量两个集合的差异，是两个集合中不同元素在并集中所占的比例，数学上 Jaccard 距离 =1-Jaccard 相似系数。

25.1.3　SAS实践

SAS 中计算相似性与距离矩阵，可以使用 PROC DISTANCE 来完成。由于一个数据集既包含分类变量又可包含定量变量，因此我们在计算变量的距离或相似性之前，需要指定变量的测量尺度。

在统计学上，测量尺度表示我们能从变量数据中获得什么信息，其包括 4 个级别：定类变量、定序变量，定距变量和定比变量；后一级的变量包含前一级变量的特性，但有自己的特性。比如定类变量只能做等于或不等于的比较，定序变量则可以做大于 / 小于的比较，但不能做加法。定距变量可以作加减运算，但不能做乘除；定比变量则不但有定距变量的间距特征，还因为存在绝对零点而可以作乘除运算。详情参阅第 18 章统计学上的变量类型一节。

SAS 对测量尺度有更加细致的考虑，它的定类变量进一步细分为非对称和对称两种，因此在 SAS 中它共包含 ANOMINAL、NOMINAL, ORDINAL、INTERVAL 和 RATIO 5 种测量尺度。

（1）前 3 个为分类变量（Categorical），包括非对称定类变量、对称定类变量和定序变量。分类变量的存储数据类型可为字符型或数值型。

（2）后两个为定量变量（Quantitative）中的定距和定比变量，其存储数据类型必须是数值型。

下面的过程步 PROC DISTANCE 计算 SASHELP.CLASS 中各观测中身高的距离，其中 EUCLID 表示计算欧氏距离（见程序 25-1）。

程序25-1　计算距离系数

```
proc distance data sashelp.class method EUCLID  out distance dist;
  var ratio( weight height);
run;
proc print;run;
```

系统会输出距离矩阵到 DISTANCE DIST 数据集中，如图 25-7 所示。

Obs	Dist1	Dist2	Dist3	Dist4	Dist5	Dist6	Dist7	Dist8	Dist9	Dist10	Dist11	Dist12	Dist13	Dist14	Dist15	Dist16	Dist17	Dist18	Dist19
1	0.0000																		
2	31.1207	0.0000																	
3	14.9646	16.5360	0.0000																
4	11.7661	19.5433	5.1478	0.0000															
5	11.4127	19.7800	4.8466	0.7000	0.0000														
6	31.7355	1.2806	17.0000	20.2608	20.4619	0.0000													
7	29.4727	3.3377	14.5774	18.2483	18.3763	2.9155	0.0000												
8	6.5000	29.1247	14.7679	10.0045	10.0499	29.9548	28.1299	0.0000											
9	29.2318	6.0000	14.2773	18.5024	18.5270	5.2953	2.7459	28.5000	0.0000										
10	16.4012	15.7003	6.4761	4.8415	5.4083	16.5873	15.0213	13.4629	15.8902	0.0000									
11	64.4771	33.9012	49.5202	53.2565	53.4120	33.0492	35.0464	63.0035	35.3227	49.6013	0.000								
12	22.9856	9.8407	8.0623	12.5897	12.5256	9.8995	7.1063	22.5719	6.2642	10.8784	41.584	0.0000							
13	37.7033	7.0029	22.8473	26.3154	26.4970	6.0828	8.2765	36.0373	9.3509	22.6614	26.968	15.2643	0.0000						
14	2.5495	29.7321	14.0513	10.1951	9.9624	30.4243	28.3044	4.0311	28.2843	14.5774	63.351	22.1097	36.4560	0.0000					
15	37.6198	67.7956	52.4299	48.3827	48.2545	68.5937	66.6265	38.6846	66.6802	52.1464	101.630	60.4921	74.6692	38.3960	0.0000				
16	16.0590	44.7760	30.0042	25.5783	25.5331	45.6207	43.7864	15.6697	44.0601	29.0842	78.667	38.0033	51.7035	16.0901	23.1482	0.0000			
17	20.5973	50.1124	35.0413	30.7878	30.7002	50.9322	49.0315	20.9881	49.2062	34.4420	83.981	43.0847	57.0131	21.0060	17.7200	5.4626	0.0000		
18	29.8077	1.4142	15.1605	18.2850	18.5000	2.0100	2.3537	27.9508	5.0990	14.5774	35.053	8.4404	8.0895	28.4605	66.5977	43.6152	48.9311	0.0000	
19	2.5495	29.7321	14.0513	10.1951	9.9624	30.4243	28.3044	4.0311	28.2843	14.5774	63.351	22.1097	36.4560	0.0000	38.3960	16.0901	21.0060	28.4605	0

图 25-7　输出距离系数

SAS 支持 39 种不同的距离和相似性系数计算方法，下面列出了 PROC DISTANCE 最常用的 METHOD 参数，如果没有特殊注明则表示该方法适用于定序、定距和定比变量。

- CITYBLOCK – 城市街区距离
- EUCLID – 欧氏距离
- CHEBYCHEV – 切比雪夫距离
- L(p) – 闵氏距离，比如L(2) 表示欧氏距离
- POWER(p,r) – 广义欧氏距离

- CANBERRA – 兰氏距离，仅适用于定比变量
- COSINE – 余弦相似系数，仅适用于定比变量
- CORR – 相关系数
- JACCARD – JACCARD相似系数，仅适用于非对称定类变量和定比变量

对于一个稍微复杂一点的数据集，需要同时指定多个变量的测量尺度，此时可以将数据集中的多个变量归入特定测量尺度并指定标准化方法，从而能完整科学地计算出距离矩阵（见程序 25-2）。

程序25-2　指定测量尺度计算距离系数

```
proc distance data sashelp.class method dgower out distance dist;
  var ratio(weight / std maxabs)
```

```
    interval(height / std=range)
    ordinal(age / std=range)run;
    nominal(sex);
run;
```

在计算相似度或距离时，可对不同分组分别进行。这时可指定分组变量，如可以按照 SEX 变量分别计算距离（见程序 25-3）。

程序25-3　按分组计算距离系数

```
proc sort data=sashelp.class out=sorttemptablesorted;
  by sex;
run;
proc distance data=sorttemptablesorted method=dgower
  out=distance_dist;
  var ratio(weight / std=maxabs);
  by sex;
run;
proc delete data=sorttemptablesorted;

proc print data=distance_dist;run;
```

系统输出结果如图 25-8 所示。

Obs	Sex	Dist1	Dist2	Dist3	Dist4	Dist5	Dist6	Dist7	Dist8	Dist9	Dist10
1	男	0.00000									
2	男	0.06667	0.00000								
3	男	0.19667	0.13000	[illegible]							
4	男	0.19000	0.12333	[illegible]	0.00000						
5	男	0.08667	0.02000	0.11000	0.10333	0.00000					
6	男	0.25000	0.31667	0.44667	0.44000	0.33667	[illegible]				
7	男	0.10333	0.17000	0.30000	0.29333	0.19000	0.14667	0.00000			
8	男	0.13667	0.20333	0.33333	0.32667	0.22333	0.11333	0.03333	0.00000		
9	男	0.18333	0.11667	0.01333	0.00667	0.09667	0.43333	0.28667	0.32000	0.00	
10	男	[illegible]	0.06333	0.19333	0.18667	0.08333	0.25333	0.10667	0.14000	0.18	0
11	女	0.00000									
12	女	0.12444	0.00000								
13	女	0.16444	0.04000	[illegible]							
14	女	0.00444	0.12000	0.16000	[illegible]						
15	女	0.25333	0.12889	0.08889	0.24889	0.00000					
16	女	0.29778	0.42222	0.46222	0.30222	0.55111	0.00000				
17	女	0.05333	0.07111	0.11111	0.04889	0.20000	0.35111	0.00000			
18	女	0.06222	0.18667	0.22667	0.06667	0.31556	0.23556	0.11556	0.00000		
19	女	0.24889	0.12444	0.08444	0.24444	0.00444	0.54667	0.19556	0.31111	0.00	
20	女										

图 25-8　分组输出距离系数

25.2　聚类形成方法

25.2.1　一次形成分类系统

从距离矩阵或相似性矩阵生成最终聚类系统有多种方法，下面以实例演示聚类的形

成过程。首先假定有如下6个数据点的数据（见程序25-4），其数据在变量空间的投点如图25-9所示。后续的分析探讨将基于此演示数据进行说明。

程序25-4 准备聚类数据

```
data d1;
  input id x y @@;
  datalines;
1 0.5 0.5 2 0.5  1.5  3 1.5 0.5
4 1.5 1.0 5 1.75 0.75 6 2.5 2.5
;

proc sgplot data=d1;
  scatter x=x y=y / datalabel=id ;
  xaxis min=0 max=3 grid  ;
  yaxis min=0 max=3 grid   ;
run;
```

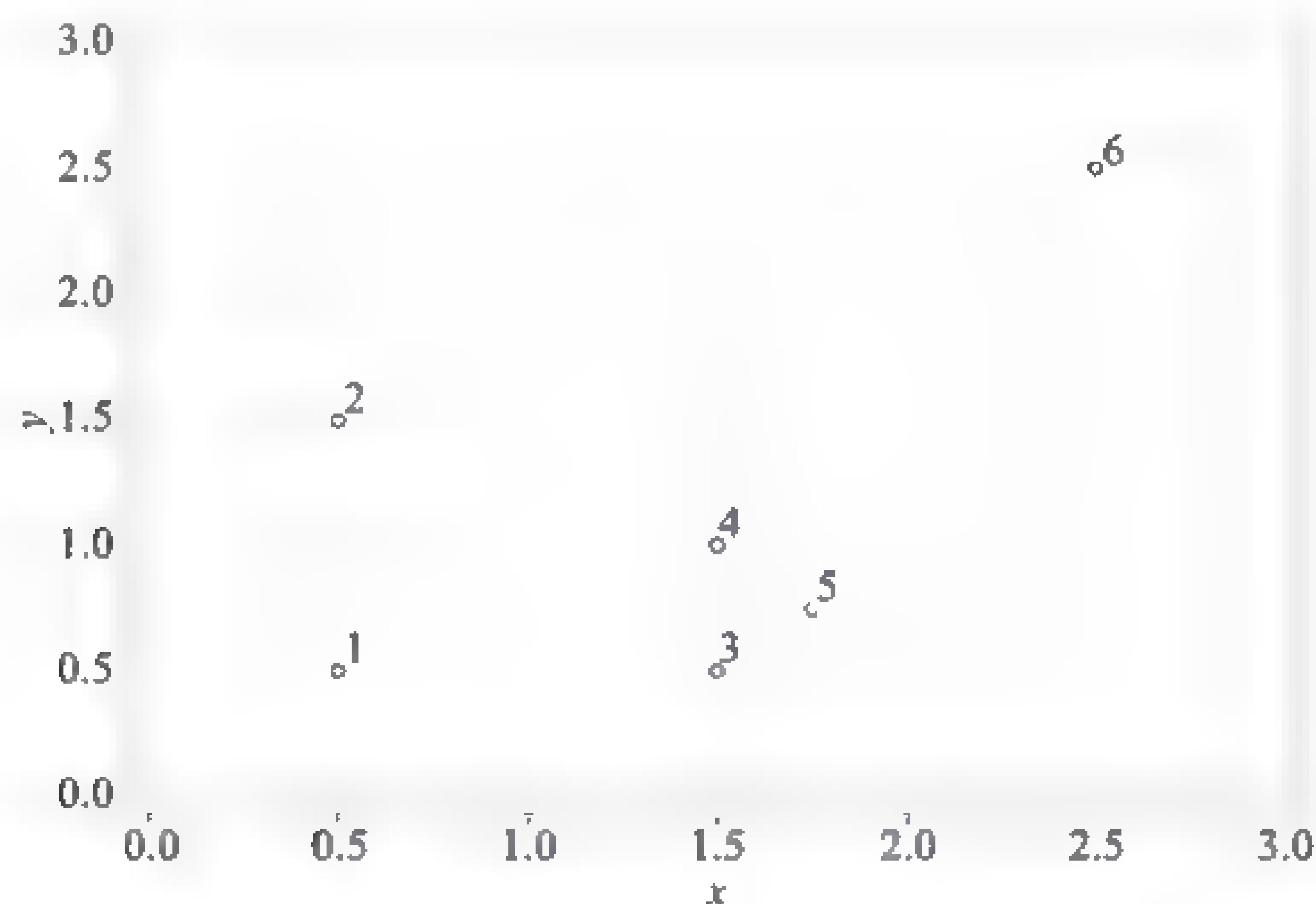


图25-9 聚类数据在二维平面的投影

现在需要对6行2列的样本数据进行聚类。一般地，对于n行m列的数据表，每列表示一个变量（也称维度或指标），每一行表示一个样本。如果聚类分析用于分析变量之间的关系，称为R型分析；如果用于对样本进行聚类，分析样本之间的关系，则称为Q型分析。理论上只要将数据进行转置，就可以使用同样的聚类分析对数据进行运算。下面以Q型聚类分析为例探讨一次性形成分类系统的聚类过程。

对于分析n行m列的样本数据$[n, m]$，可对n个样本两两之间进行尺度计算，形成$[n, n]$的方形尺度矩阵，且该矩阵上下三角对称。如果采用距离作尺度，对角线元素为0，其余矩阵元素大于零，数值越大表示距离越远；如果是相似性/相关系数矩阵，对角线元素为1，其余矩阵元素取值在[-1, 1]之间，正负号表示正相关还是负相关，绝对值越大表示相似性/相关性越强。

对于前面的演示数据，可以根据尺度计算方法很容易算出欧氏距离方阵，其中计算距离前使用了Min-Max标准化处理（请参考数据标准化部分）。对应的SAS代码如程序25-5所示，输出结果如图25-10所示。

程序25-5 计算距离数据

```
proc distance data d1 method EUCLID out d1 dist;
```

```
  var ratio( x y /std range ) ;
run;
```

Obs	Dist1	Dist2	Dist3	Dist4	Dist5	Dist6
1	0.00000					
2	0.50000	0.00000				
3	0.50000	0.70711	0.00000			
4	0.55902	0.55902	0.25000	0.00000		
5	0.63738	0.72887	0.17678	0.17678	0.00000	
6	1.41421	1.11803	1.11803	0.90139	0.95197	0

图 25-10　聚类数据

该矩阵揭示了 6 个样本两两之间的距离，一次形成聚类的方法就是直接从上三角矩阵中按照距离从小到大（如果是相似性，则从大到小）的顺序构建聚类树的过程。聚类时从该方阵上三角矩阵中从小到大依次寻找最短距离，并剔除对应的列，则依次搜索结果为 [3, 5]=0.177，[3, 4]=0.250，[1, 2]=0.500，[1, 3]=0.500，[4, 6]=0.901。因此其聚类谱系图的形成步骤为：创建新类 [3, 5]，合并 4 到 3 所在类，创建新类 [1, 2]，合并 1 到 3 所在类，最后合并 6 到 4 所在聚类（见图 25-11）。

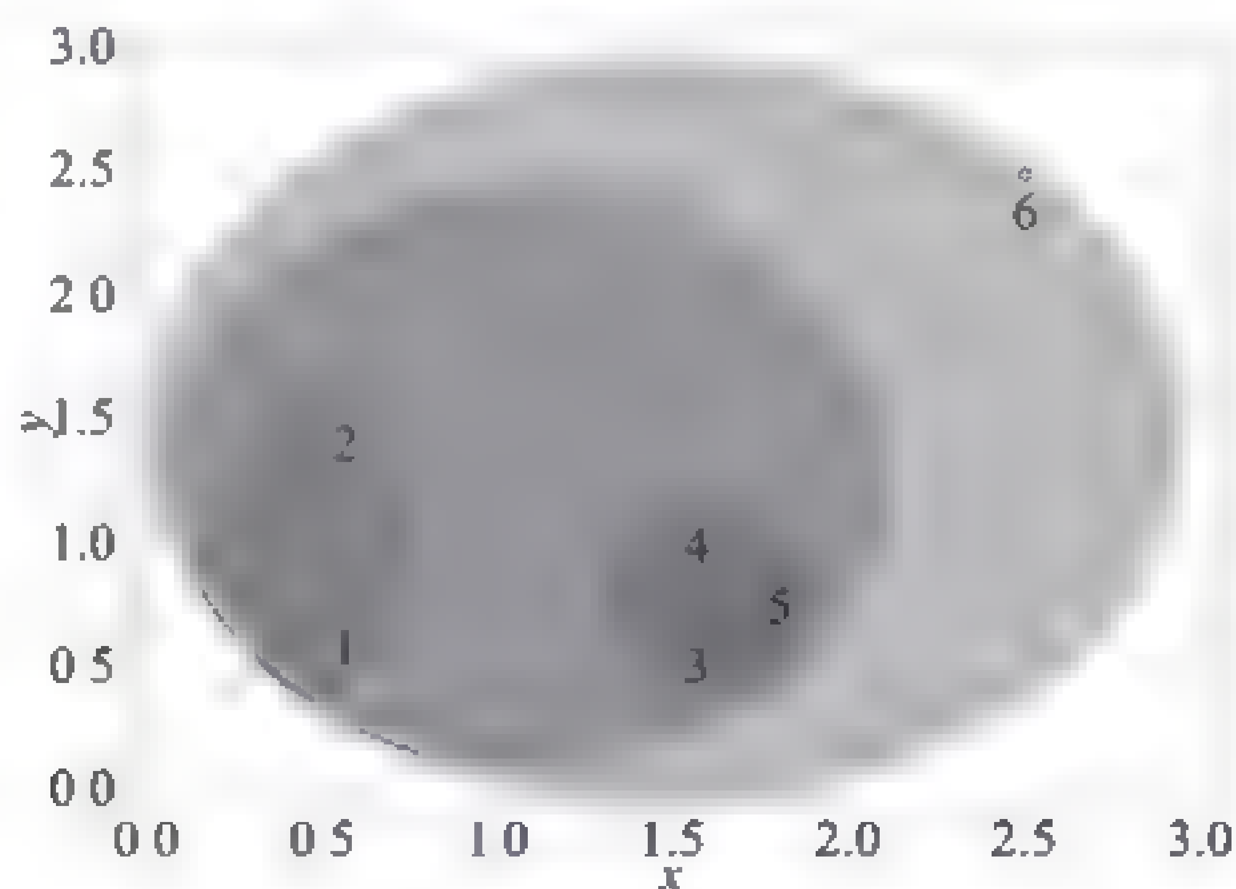


图 25-11　一次形成分类系统结果

25.2.2　K-均值聚类

K- 均值聚类的思想最早是 Stuart Lioyd 在 1957 年提出的，而 E. W. Forgy 在 1965 年也发表了本质上一样的思想方法，只不过那个时候并不叫 K- 均值而已。英文 “K-means” 一词直到 1967 年才由 James MacQueen 提出，K- 均值方法有时也叫劳埃德方法或者 Lioyd-Forgy 方法。

K- 均值聚类的核心思想为指定划分数目的最佳划分。对于 n 个观测，每个观测是一个 m 维的实数向量，现在需要找到 k 个聚类（其中 $k \leqslant n$，即 k 个子集），使得每个类别分组内的方差最小化，因此 K- 均值的目的就是从 n 个观测中找到 k 个 m 维向量，使下面的等式成立：

$$\arg_S \min \sum_{i=1}^{k} \sum_{x \in S_i} \left\| x - \mu_i \right\|^2 = \arg_S \min \sum_{i=1}^{k} |S_i| Var S_i$$

式中：$S=\{S_1, S_1, \cdots, S_k\}$ 为 k 个子集；μ_i 为第 i 个子集所有数据的均值（或称质心）。由于数据的总方差是恒定的，如果一个子集内的方差达到最小化，则意味着子集间所有点之间的方差达到最大化，即组内差异尽可能小，组间差异尽可能大。如果将分配类别和更新质心两阶段分别被看作期望计算阶段（E）和最大化阶段（M），则 K- 均值方法在广义上是期望最大化（Expectation Maximization，EM）算法的一个变体。

K- 均值聚类的核心实现步骤为以下内容。

（1）随机选取 k 个真实 / 或虚拟的数据点作为初始质心。初始质心的选择对聚类结果具有重要影响，因此如何科学选择这些初始质心很重要，一般选择彼此尽可能远的若干样本。

（2）分配类别阶段：计算每一个观测到这 k 个聚类中心的距离，将观测唯一分配给距离最近的那个质心，从而将每个观测分配到 k 类中的一个。

（3）更新质心阶段：根据观测所属类别，重新计算 k 个类别的均值，即更新 k 个聚类中心。

（4）反复（2）、（3）两个步骤，直到聚类中心不再发生改变，迭代达到指定次数或改进小于指定阈值为止。

K- 均值聚类算法的核心数据结构和自我迭代收敛如图 25-12 所示，分配类别和更新质心两个阶段反复执行，直到迭代收敛为止。

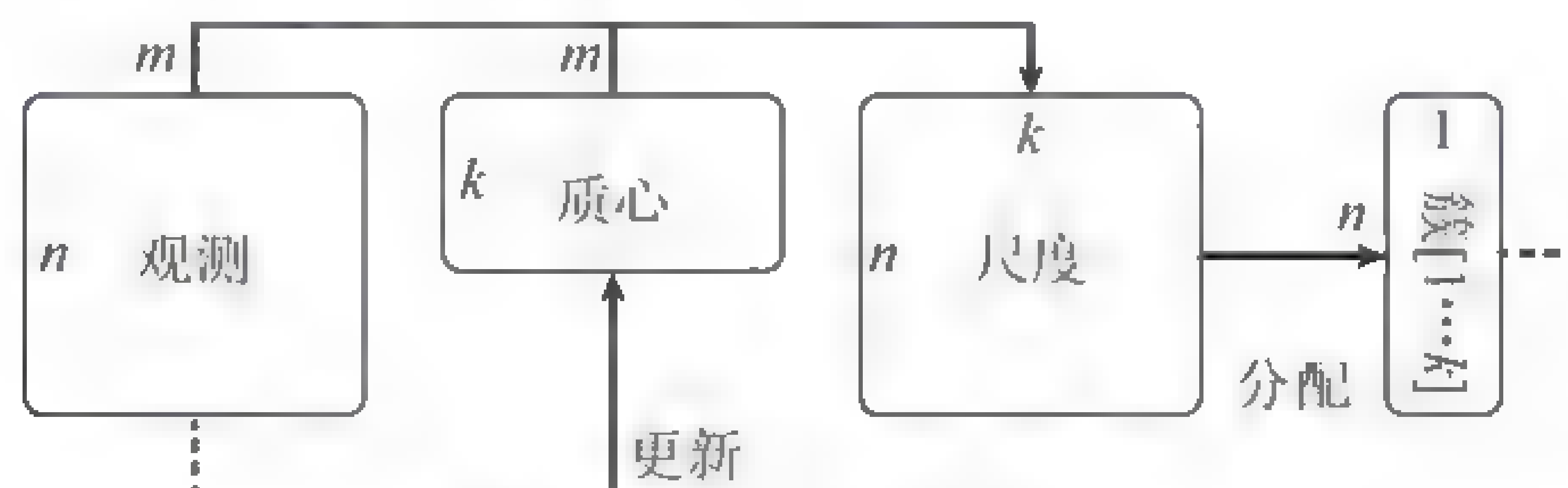


图 25-12　K- 均值迭代过程

为进一步展示实际运行过程，对于前面 6 个演示数据点进行迭代。为直观演示目的，数据未作 Min-Max 标准化。假定初始值质心就是数据中的前 3 个点：(0.500, 0.500), (0.500, 1.500), (1.500, 0.500)，则 K- 均值聚类形成中每一步的关键数据结构如表 25-1 所列，每个观测到 3 个质心的距离在尺度的 3 列中显示。

表 25-1　K- 均值计算过程

观测 [n, m]	质心 [k, m]	尺度 [n,k]	聚 类 分 配
0.500 0.500	0.500 0.500	**0.000** 1.000 1.000	Obs[1]->Cluster[**1**] *
0.500 1.500	0.500 1.500	1.000 **0.000** 1.414	Obs[2]->Cluster[**2**] *
1.500 0.500	1.500 0.500	1.000 1.414 **0.000**	Obs[3]->Cluster[**3**] *
1.500 1.000		1.118 1.118 **0.500**	Obs[4]->Cluster[**3**] *
1.750 0.750		1.275 1.458 **0.354**	Obs[5]->Cluster[**3**] *
2.500 2.500		2.828 **2.236** 2.236	Obs[6]->Cluster[**2**] *

（续表）

观测 [n, m]	质心 [k, m]	尺度 [n,k]	聚类分配
0.500 0.500	0.500 0.500	**0.000** 1.803 1.112	Obs[1]->Cluster[1]
0.500 1.500	1.500 2.000	**1.000** 1.118 1.318	Obs[2]->Cluster[1] *
1.500 0.500	1.583 0.750	1.000 1.500 **0.264**	Obs[3]->Cluster[3]
1.500 1.000		1.118 1.000 **0.264**	Obs[4]->Cluster[3]
1.750 0.750		1.275 1.275 **0.167**	Obs[5]->Cluster[3]
2.500 2.500		2.828 **1.118** 1.976	Obs[6]->Cluster[2]
0.500 0.500	0.500 1.000	**0.500** 2.828 1.112	Obs[1]->Cluster[1]
0.500 1.500	2.500 2.500	**0.500** 2.236 1.318	Obs[2]->Cluster[1]
1.500 0.500	1.583 0.750	1.118 2.236 **0.264**	Obs[3]->Cluster[3]
1.500 1.000		1.000 1.803 **0.264**	Obs[4]->Cluster[3]
1.750 0.750		1.275 1.904 **0.167**	Obs[5]->Cluster[3]
2.500 2.500		2.500 **0.000** 1.976	Obs[6]->Cluster[2]

图 25-13 可更直观地看到 K- 均值聚类的生成过程，对 6 个点的聚类只用了 2 次迭代即收敛。小圆圈为初始质心和每一步聚类后更新的质心，椭圆或大圆圈为数据根据已有质心形成的分类。

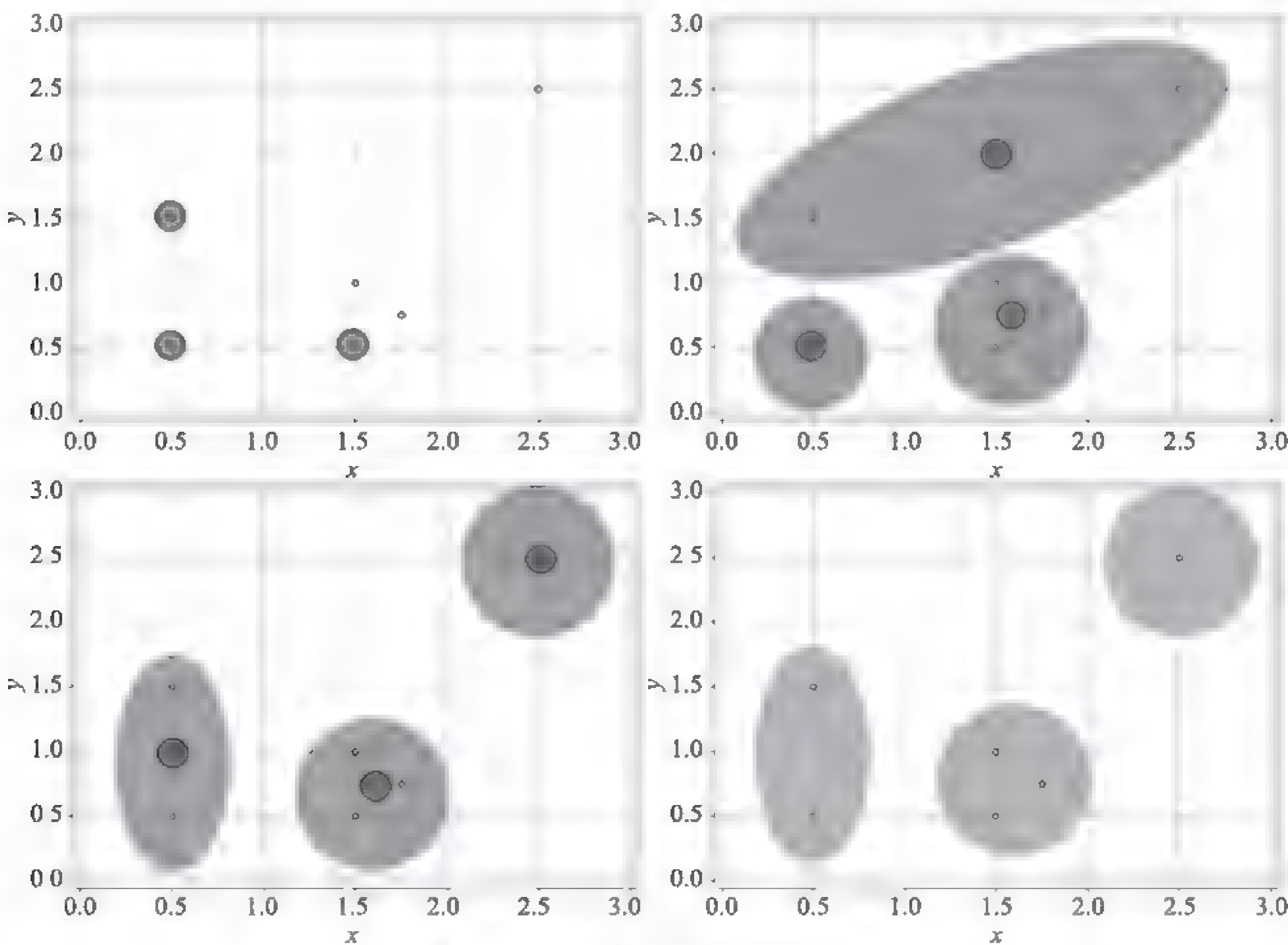


图 25-13　K- 均值形成聚类过程

K- 均值聚类算法主要有如下一些特征。

（1）需要预先指定分类数目 k 或初始质心 $c[k, m]$ 信息。如果 k 值或初始质心选不恰当可能会产生不好的聚类，因此需要反复调试确定最优分类数目。

（2）使用欧氏距离作为分类度量，方差作为组间差异的度量。如果使用欧氏距离以外的其他距离函数，需要确保迭代算法是收敛的，即迭代能够确保组内方差与上一次比减少。

（3）该算法可收敛到局部最优而非全局最优，因此某些情况下（如初始质心选取不当）它可能导致一些违反直觉的聚类结果。比如图25-14从左到右迭代过程并不能拉开左上角的两个聚类中心的距离，而左下角的聚类则明显显得过于庞大。

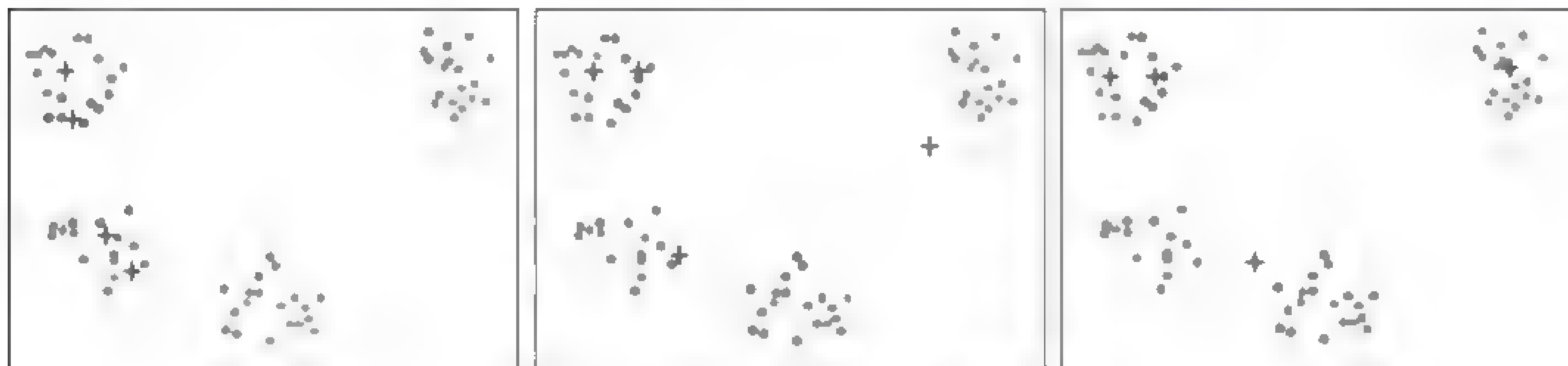


图25-14 初始质心选取会影响结果

K-均值算法在SAS里的实现为PROC FASTCLUS过程步，它基于欧氏距离和最小二乘估计聚类中心，自动寻找最佳初始质心来将观测分成不同的子集。K-均值算法并不形成树状聚类结构，而是划分为若干平等的子集。K-均值算法还可用来检查离群值，因为离群值通常自成一类而不是跟别的观测聚集成簇。

除了基于最小二乘法，PROC FASTCLUS也可以用LEAST=P选项指定最小概率法。与最小二乘法相比，一般P值小于2能降低异常值对集群中心的影响，而P值大于2则会增加异常值的影响。

PROC FASTCLUS其系统耗时与样本量n成正比，因此它是一种快速聚类方法，可用于处理较大规模的数据集。实践中一般将它用于较大规模数据的预先处理形成分区，然后使用PROC CLUSTER再次聚类。但对于小数据量的数据，聚类结果可能因初始质心的选择而对样本出现顺序敏感。而且由于K-均值方法受数值较大的变量影响更大，通常需要对样本数据先作标准化处理。

程序25-6对演示数据进行K-均值聚类，其中K值由最大聚类数maxclusters值指定。用户也可指定选择新种子的最小距离radius来执行聚类。

程序25-6 SAS 执行K-均值聚类

```
proc fastclus data=d1 maxclusters=3;
  id id;
run;
```

或

```
proc fastclus data=d1 radius=1;
  id id;
run;
```

SAS系统输出如图25-15所示。

从图25-15输出中可以看到，系统选择最远的3个点(2.5, 2.5), (0.5, 1.5), (1.75, 0.75),

即编号为 6，2，5 的 3 个点作为初始种子（质心），并最终形成了 3 个聚类（见图 25-16），其质心分别为 (2.5, 2.5), (0.5, 1.0) 和 (1.58, 0.75)。

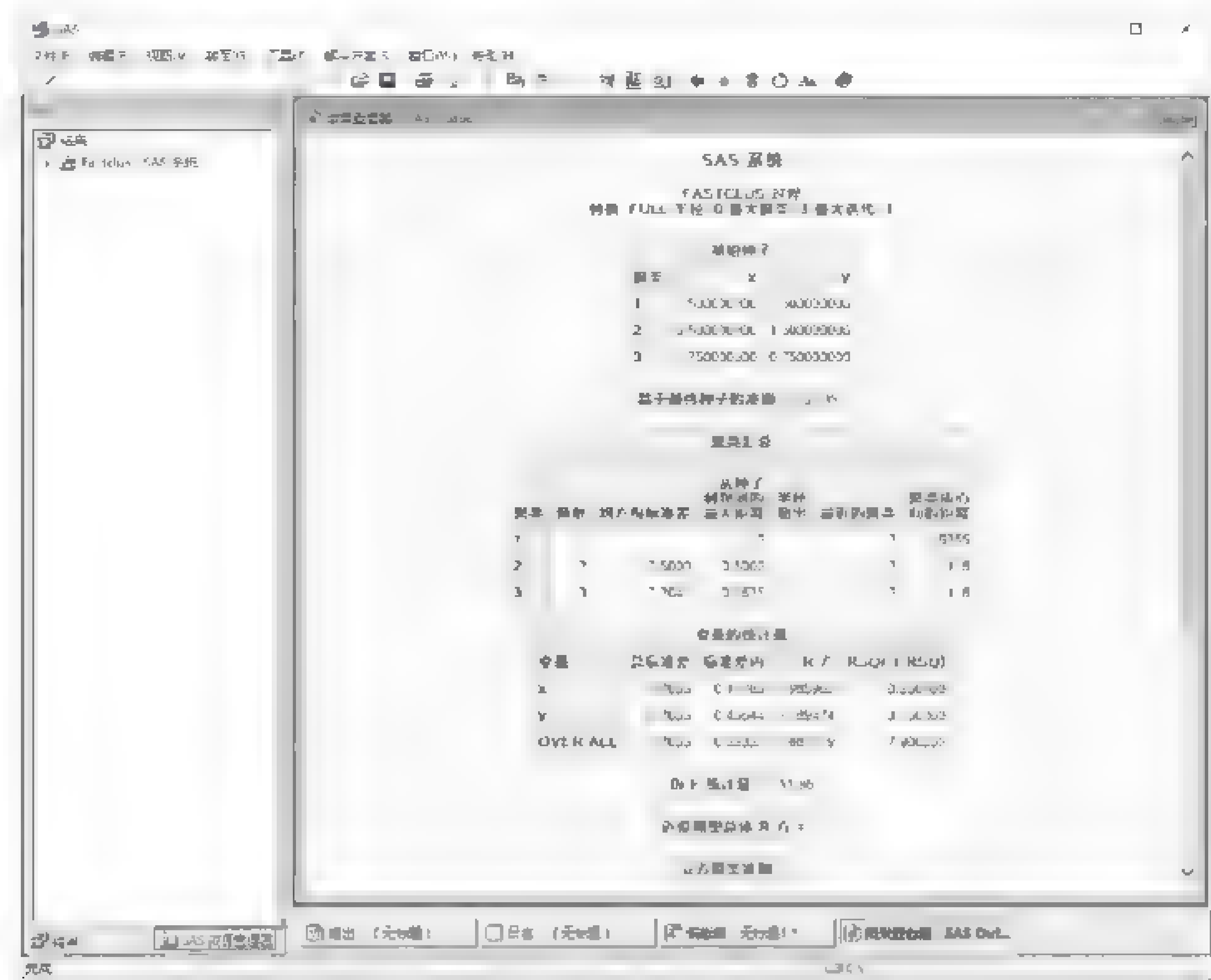


图 25-15　FastClus 聚类结果

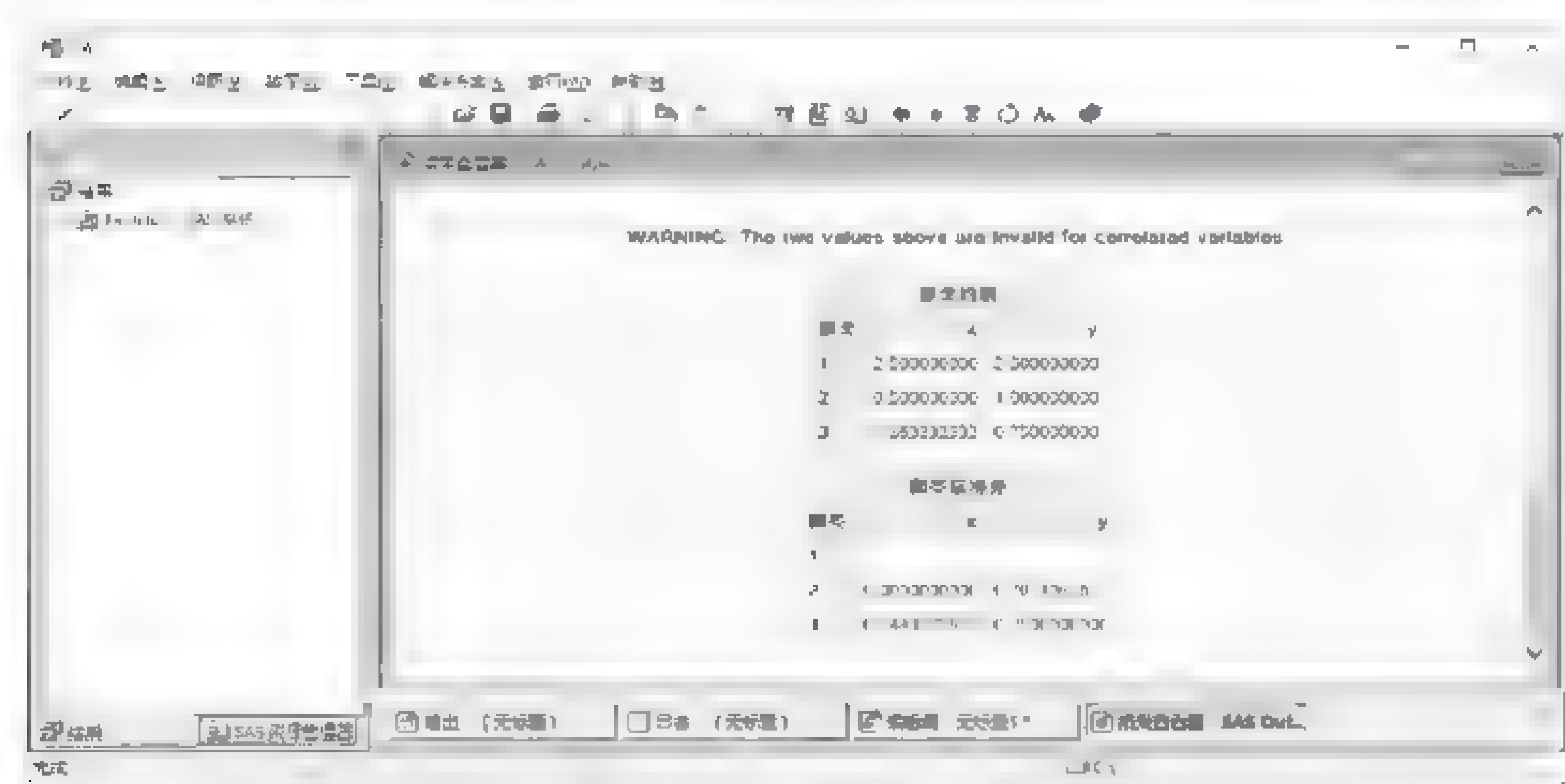


图 25-16　聚类均值

为了显示每个样本的归类，可以使用如下方法来查看各聚类的详细信息（见程序 25-7）。首先使用 OUT 选项将 PROC FASTCLUS 的结果输出到数据集（见图 25-17），然后再使用 PROC FREQ 过程步查看频数。

程序25-7　输出聚类详情

```
proc fastclus data=d1 maxclusters=3 out=d1_cls;
  id id;
run;

proc freq data=d1_cls;;
  tables cluster*id;
run;
```

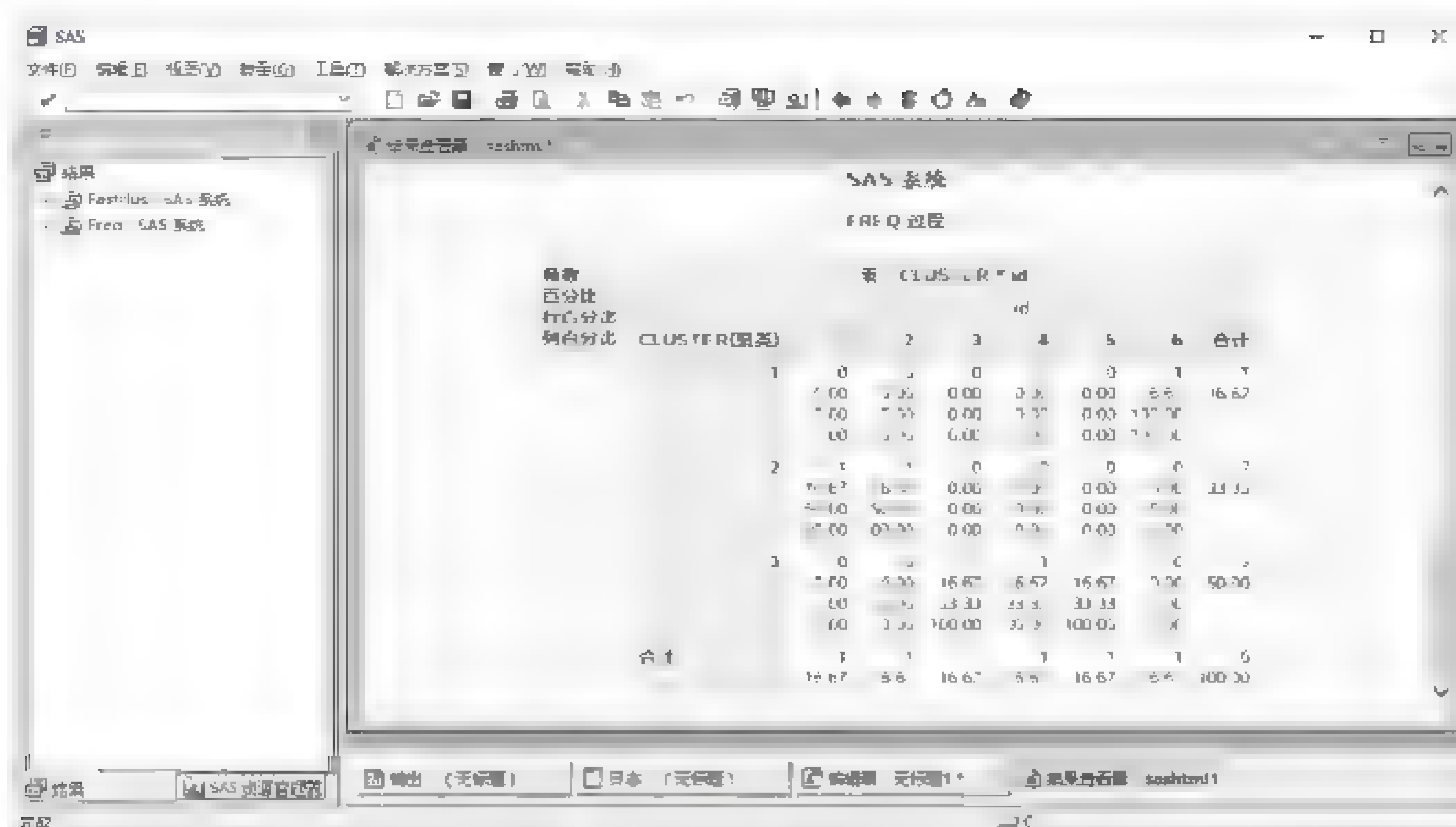


图 25-17 聚类细节

从图 25-17 中可以看到聚类 1 包含 6 号样本，聚类 2 包括 1 号和 2 号 两个样本，聚类 3 包括 3 号、4 号、5 号三个样本。为了更加直观地看它们的分类情况，可以使用程序 25-8 的 PROC SGPLOT 来投影，其中聚类分组 CLUSTER 被使用在 SCATTER 语句的 GROUP 选项上对不同组用不同形状的标记进行区分，结果如图 25-18 所示。

程序25-8 K-means 结果分组投影

```
data attr_map;/*创建属性表*/
  input id $ value $ fillcolor $ markercolor $ markersymbol $ ;
  datalines;
id1 1 CXDE6D00 CXDE6D00 circle
id1 2 CX00537F CX00537F square
id1 3 CXc0c0c0 CX00537F star
;
run;
proc sgplot data=d1_cls dattrmap=attr_map;/*用属性表进行分组*/
  scatter y=y x=x / group=Cluster attrid=id1;
  xaxis min=0 max=3 grid;
  yaxis min=0 max=3 grid;
run;
```

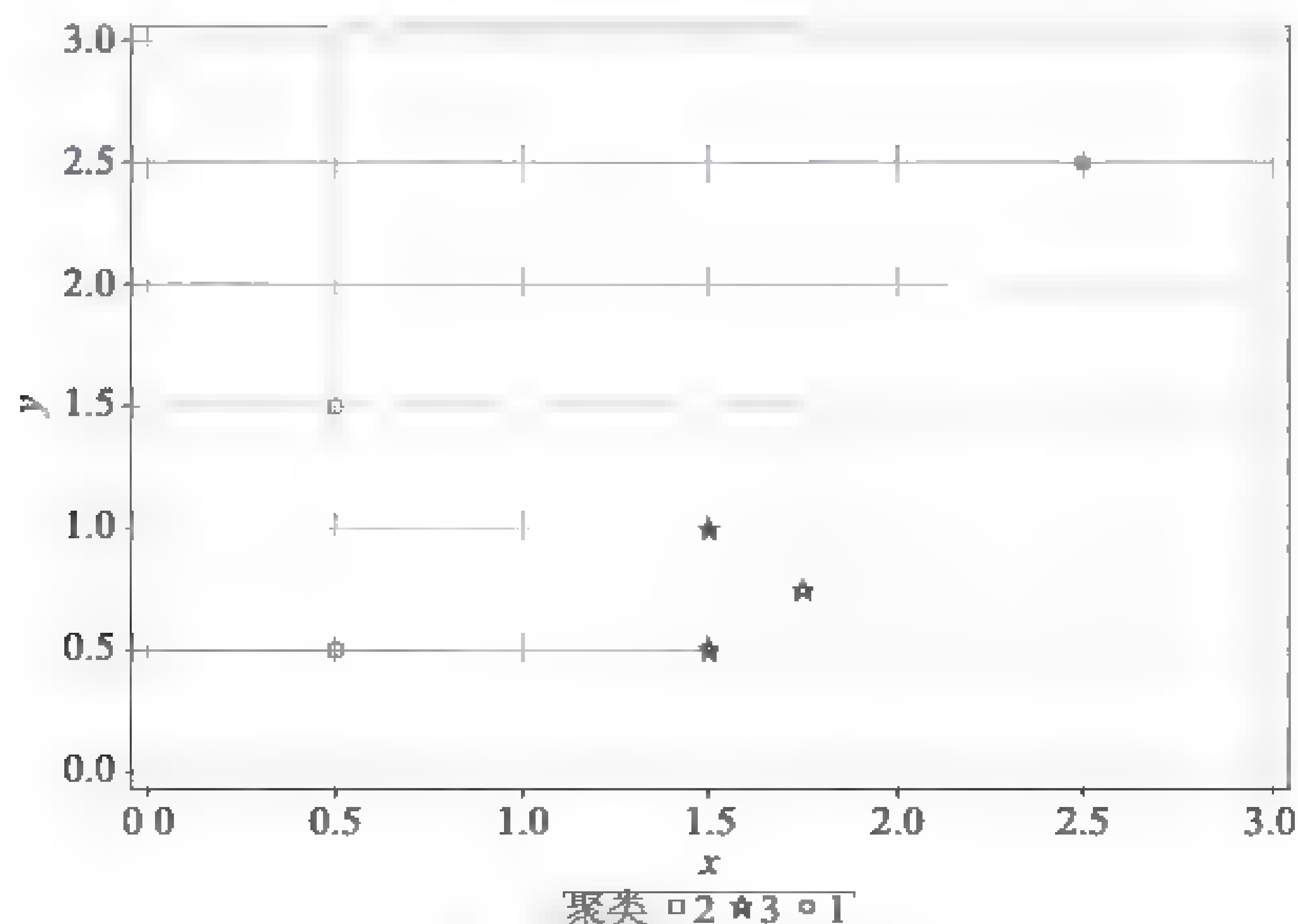


图 25-18 分组投影

如果待分析的数据不仅仅包括 x，y 两个维度而是包括三个以上的维度，则可以使用典型判别分析来进行降维，然后再将结果投影在两个主要指标上来绘制最佳的划分图像。典型判别分析是跟主成分分析和典型相关分析有关的降维技术，完整代码如程序 25-9 所示，结果输出如图 25-19 所示。

```
程序25-9  以SASHELP.CLASS 为例用典型判别来投影
proc fastclus data=sashelp.class maxclusters=4 out=class_cluster;
  id name;
run;

/*对 K-均值聚类结果进行典型判别分析*/
proc candisc data=class_cluster anova out=class_cluster_can;
  class cluster;
  var age height weight  ;
run;

data attr_map;/*创建属性表*/
  input id $ value $ fillcolor $ markercolor $ markersymbol $ ;
  datalines;
id1 1 CXDE6D00 CXDE6D00 circle
id1 2 CX00537F CX00537F square
id1 3 CXc0c0c0 CX00535F star
id1 4 CXc0c0c0 CX00532F triangle
;
run;
/*将聚类投影在两个主要主要指标上*/
proc sgplot data=class_cluster_can dattrmap=attr_map;
  scatter x=Can1 y=Can2 / group=Cluster attrid=id1 datalabel=name;
run;
```

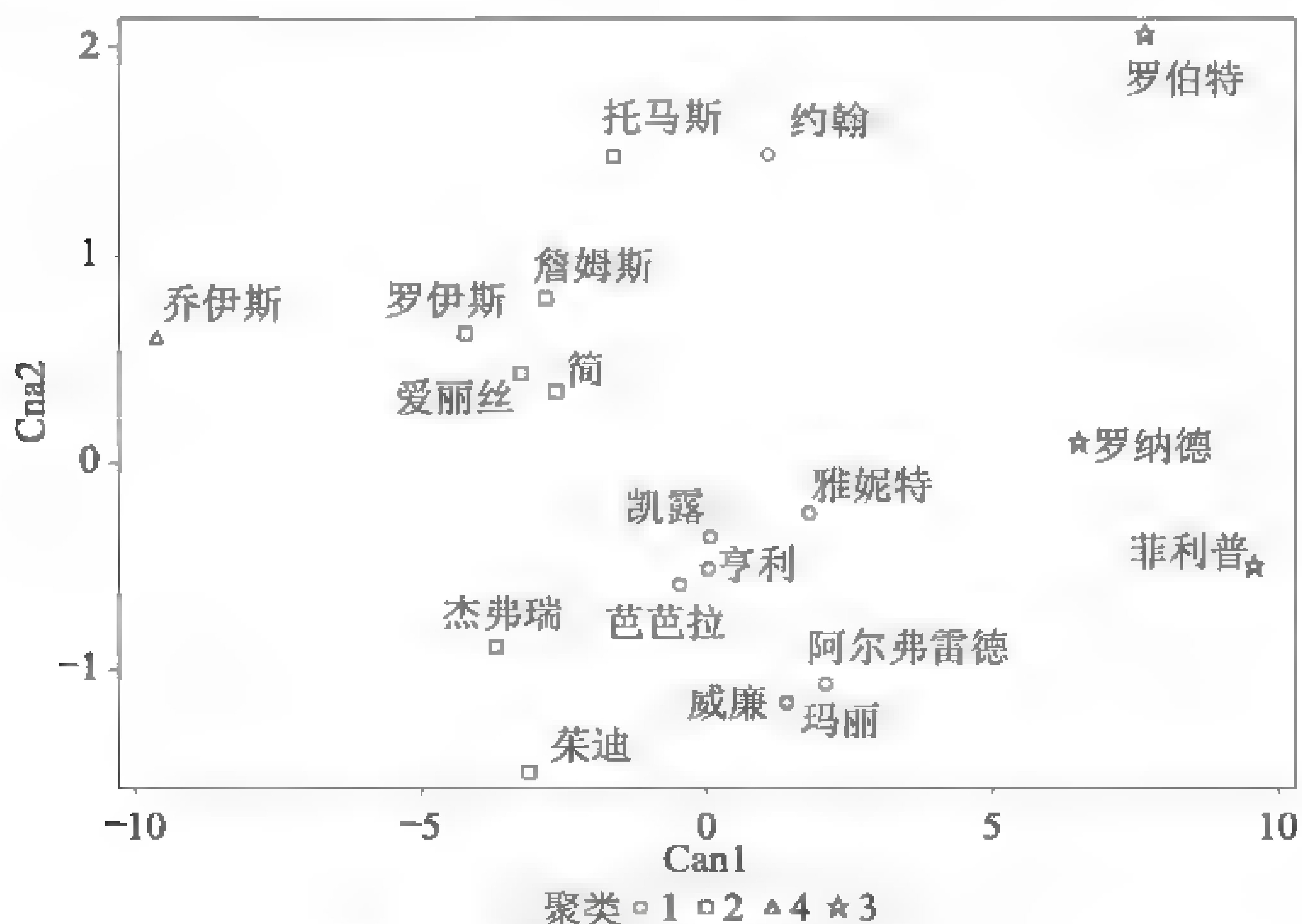


图 25-19 检查聚类效果

在实践中一般需要对数据进行标准化后才进行聚类处理。SAS 提供 PROC STDIZE 完成对应的工作（见程序 25-10）。

程序25-10 标准化后与K均值聚类

```
proc stdize data=sashelp.class out=class_std method=range;
  var weight height ;
run;

proc fastclus data=class_std maxclusters=4;
  var weight height;
run;

proc delete data=class_std;
run;
```

如要数据集中同时包含不同的组，只需要简单引入分析分组变量即可（见程序25-11），如按照性别分别进行聚类分析，系统输出如图25-20所示。

程序25-11 按组进行K均值聚类

```
proc sort data=sashelp.class out=data_sort;
  by sex;
run;
proc stdize data=data_sort out=data_std method=range;
  var weight height;
  by sex;
run;
proc fastclus data=data_std maxclusters=4 out=data_cluster;
  var weight height;
  by sex;
run;
proc delete data=data_sort;
run;
proc delete data=data_std;
run;
proc print data=data_cluster;
  by sex;
run;
```

性别=男

Obs	Name	Age	Height	Weight	CLUSTER	DISTANCE
1	阿尔弗雷德	14	0.79592	0.44030	1	0.22769
2	亨利	14	0.42177	0.29104	2	0.06373
3	詹姆斯	12	0.00000	0.00000	3	0.10162
4	杰弗瑞	13	0.35374	0.01493	2	0.25763
5	约翰	12	0.11565	0.24627	3	0.17045
6	菲利普	16	1.00000	1.00000	4	0.00000
7	罗伯特	12	0.51020	0.67164	1	0.15424
8	罗纳德	15	0.65986	0.74627	1	0.12695
9	托马斯	11	0.01361	0.02985	3	0.06882
10	威廉	15	0.62585	0.43284	1	0.18888

性别=女

Obs	Name	Age	Height	Weight	CLUSTER	DISTANCE
11	爱丽丝	13	0.34211	0.54032	1	0.07644
12	芭芭拉	13	0.92105	0.76613	2	0.07903
13	凯蕾	14	0.75658	0.83871	2	0.12668
14	简	12	0.55921	0.54839	1	0.15520
15	珍妮特	15	0.73684	1.00000	3	0.13164
16	乔伊斯	11	0.00000	0.00000	4	0.00000
17	茱迪	14	0.85526	0.63710	2	0.11076
18	罗伊斯	12	0.32895	0.42742	1	0.11252
19	玛丽	15	1.00000	0.99194	3	0.13164

图25-20 分组K-均值聚类

25.2.3　逐步形成分类系统

逐步形成分类系统在数据标准化和尺度计算上跟一次形成分类系统本质上没有什么不同，但逐步形成分类系统在尺度矩阵中找到关系最近的一对进行归类后，它需要将这两个样本进行加权平均，作为一个虚拟的“新样本”与其他尚未进行聚类的样本一起进行后续聚类，直到所有的样本都被归类为止。

在代码实践中通常也会定义一个权重矩阵，如两个样本 i 和 j 聚类后，设置 i 的权重为 2，j 的权重为 0，权重为 0 的样本不再参与后续的分类过程，从而保持尺度矩阵的重用性。表 25-2 列出了基于 Min-Max 标准化，采用欧氏距离逐步形成分类系统的完整过程。结果可看出它跟一次形成分类系统结果基本一样，但样本之间的距离与逐步合并样本计算稍有不同。

表 25-2　逐步形成分类系统

Min-Max 标准化	欧氏距离矩阵	聚类选择	权重
0.000　0.000	0.000　0.500　0.500　0.559　0.637　1.414	[3,5]=0.177	1
0.000　0.500	0.500　0.000　0.707　0.559　0.729　1.118		1
0.500　0.000	0.500　0.707　0.000　0.250　**0.177**　1.118		2
0.500　0.250	0.559　0.559　0.250　0.000　0.177　0.901		1
0.625　0.125	0.637　0.729　0.177　0.177　0.000　0.952		0
1.000　1.000	1.414　1.118　1.118　0.901　0.952　0.000		1
0.000　0.000	0.000　0.500　0.566　0.559　0.637　1.414	[3,4]=0.198	1
0.000　0.500	0.500　0.000　0.713　0.559　0.729　1.118		1
0.563　0.063	0.566　0.713　0.000　**0.198**　0.088　1.035		3
0.500　0.250	0.559　0.559　0.198　0.000　0.177　0.901		0
0.625　0.125	0.637　0.729　0.088　0.177　0.000　0.952		0
1.000　1.000	1.414　1.118　1.035　0.901　0.952　0.000		1
0.000　0.000	0.000　**0.500**　0.556　0.559　0.637　1.414	[1,2]=0.500	2
0.000　0.500	0.500　0.000　0.659　0.559　0.729　1.118		0
0.542　0.125	0.556　0.659　0.000　0.132　0.083　0.988		3
0.500　0.250	0.559　0.559　0.132　0.000　0.177　0.901		0
0.625　0.125	0.637　0.729　0.083　0.177　0.000　0.952		0
1.000　1.000	1.414　1.118　0.988　0.901　0.952　0.000		1
0.000　0.143	0.000　0.286　**0.560**　0.500　0.641　1.317	[1,3]=0.560	5
0.000　0.429	0.286　0.000　0.691　0.576　0.758　1.152		0
0.542　0.000	0.560　0.691　0.000　0.149　0.083　1.100		0
0.500　0.143	0.500　0.576　0.149　0.000　0.190　0.992		0
0.625　0.000	0.641　0.758　0.083　0.190　0.000　1.068		0
1.000　1.000	1.317　1.152　1.100　0.992　1.068　0.000		1
0.325　0.057	0.000　0.494　0.224　0.195　0.305　1.160	[4,6]=0.992	5
0.000　0.429	0.494　0.000　0.691　0.576　0.758　1.152		0
0.542　0.000	0.224　0.691　0.000　0.149　0.083　1.100		0
0.500　0.143	0.195　0.576　0.149　0.000　0.190　**0.992**		1
0.625　0.000	0.305　0.758　0.083　0.190　0.000　1.068		0
1.000　1.000	1.160　1.152　1.100　0.992　1.068　0.000		0

逐步形成分类系统大多数是自下而上的不断聚合过程，最具代表性方法包括 CURE、ROCK、BIRCH 和 CHAMELEON 方法：CURE 方法采用了抽样和分区技术，选择数据空间中固定数目的代表性点来代表相应的类，能识别复杂形状和不同大小的聚类；ROCK 方法是改进 CURE 以适用于类别型数据的方法；KARYPIS 等人 1999 年提出的 CHAMELEON 方法则是同时考虑聚类之间连通性和相似性的两阶段层次聚类算法，聚类过程中使用了动态建模技术并能够发现任意大小和形状的聚类。

层次聚类在 SAS 中的实现是 PROC CLUSTER 过程步，它一开始假定每个样本自成一类，然后不断融合直到全部样本合并为同一个类为止。层次聚类的耗时与样本量平方或立方成正比，因此对较大规模的数据量不太适用。PROC CLUSTER 的输入数据可以是坐标或者距离，也可使用 PROC DISTANCE 过程步先行计算将距离作为 PROC CLUSTER 的输入。

PROC CLUSTER 过程步可使用 OUTTREE= 选项指定输出数据集，然后再用 PROC TREE 过程步的 NCLUSTERS= 选项来确定聚类的阈值以输出聚类成员。

PROC CLUSTER 过程步提供 11 种分层聚类方法，只要用 METHOD= 选项指定如下任一参数值即可：CENTROID、AVERAGE、SINGLE、COMPLETE、MEDIAN、FLEXIBLE、MCQUITTY、WARD、EML、DENSITY 和 TWOSTAGE，其中 DENSITY 方法还包括 3 个子方法：第 K- 最近邻方法（需用 K= 参数指定）、均匀密度方法（需用 R= 参数指定）和王氏混合法（用 HYBRID 指定）。

在自下而上的层次聚类可通过存储数据、存储距离或者是排序距离 3 种不同算法来实现凝聚层次聚类。在 SAS 中 PROC CLUSTER 的实现情况如表 25-3 所列，所有的层次聚类方法都支持使用坐标数据进行聚类，而要求使用存储距离或者排序距离的方法会自动从坐标数据计算出距离来。图 25-21 中 CENTROID、AVERAGE 和 WARD 方法可使用存储数据或存储距离两种算法：默认使用存储数据的算法进行聚类，若输入数据是距离数据，或用户指定了 NOSQUARE 选项，则 PROC CLUSTER 使用基于存储距离算法，而最后两种聚类方法 DENSITY 和 TWOSTAGE 只能用排序距离算法。

表 25-3 实现凝聚层次聚类的 3 种算法

聚类方法	描述	存储数据	存储距离	排序距离
EML	对数似然法	√		
WARD	WARD 最小方差法（离差平方和法）	√	√	
CENTROID	质心	√	√	
AVERAGE	平均连接（类平均法）	√	√	
SINGLE	单连接		√	
COMPLETE	完全连接		√	
MEDIAN	中值方法（中间距离法）		√	
FLEXIBLE	可变类平均法		√	
MCQUITTY	McQuitty 相似度		√	
DENSITY	密度连接（包括 K- 最近邻方法）			√
TWOSTAGE	两阶段密度连接			√

下面对前面演示数据集 d1 分别用 11 种方法进行聚类，从结果图中可以看出 EML 对数似然法和 WARD 最小方差法类似，而 Density 密度连接法和 TwoStage 两阶段密度连接法结果类似，这两类方法输出跟其他聚类方法输出结果不同，主要差别表现在编号 3、4 以及编号 4、5 的聚类形成上。演示数据集 d1 的 3 种不同聚类结果如图 25-21 所示。

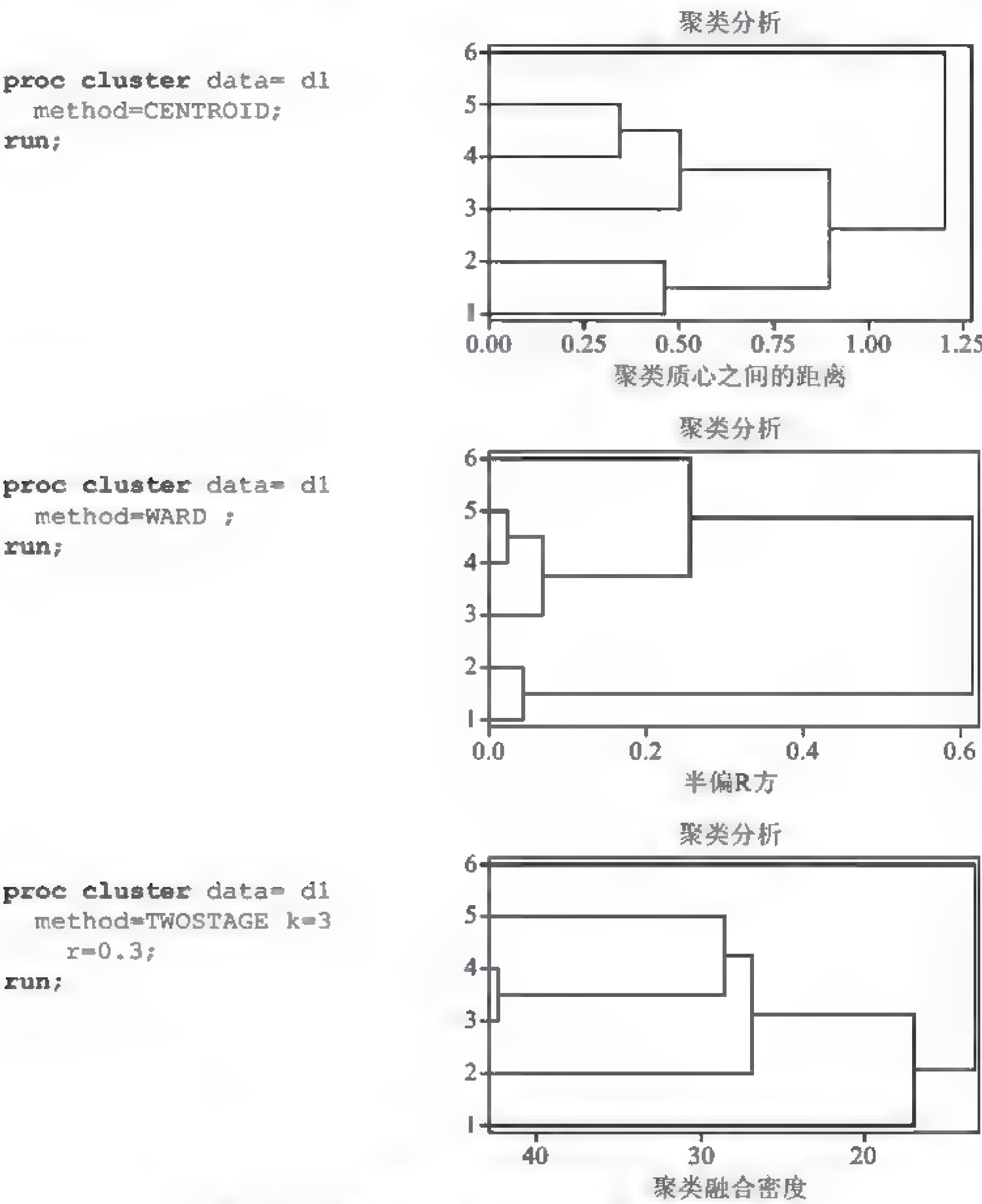


图 25-21 数据集 d1 的 3 种不同聚类结果

对于聚集层次聚类生成的结果，也可以用主成分分析来生成两个主要分量，然后将聚类结果投影在两个主分量的坐标空间中，来检查数据点彼此的分离程度和判断聚类是否足够有效。程序 25-12 运行的结果如图 25-22 所示，不同的投点标记指示了不同的类别。

程序25-12 聚类分析结果投影到主分量空间

```
/* 1)用【平均连接法】聚类并输出谱系图*/
proc cluster data=sashelp.class method=average outtree=class_tree;
  var age weight height;
  id name;
run;
```

```
/* 2) 聚类结果可以使用 proc Tree 进一步裁剪(此时才有 cluster 列)和绘图*/
proc tree data class tree out class cluster nclusters 4;
  id name;
run;

/* 3) 主成分分析找到3个主分量 prin1,prin2,prin3 */
proc princomp data=sashelp.class out=class princomp;
  var  age weight height;
run;

/* 4) 合并主成分分析和聚类分析的输出数据，按名字合并*/
proc sort data=class_princomp;
  by name;
run;
proc sort data=class_cluster;
  by name;
run;
data class_merged;
  merge class_princomp class_cluster;
  by name;
run;

/* 5) 将聚类投影在两个主要主要分量上*/
data attr_map;/*创建属性表*/
  input id $ value $ fillcolor $ markercolor $ markersymbol $ ;
  datalines;
id1 1 CXDE6D00 CXDE6D00 circle
id1 2 CX00537F CX00537F square
id1 3 CXc0c0c0 CX00535F star
id1 4 CXc0c0c0 CX00532F triangle
;
run;
proc sgplot data=class_merged dattrmap=attr_map;
  scatter x=Prin1 y=Prin2 / group=Cluster attrid=id1 datalabel=name;
run;
```

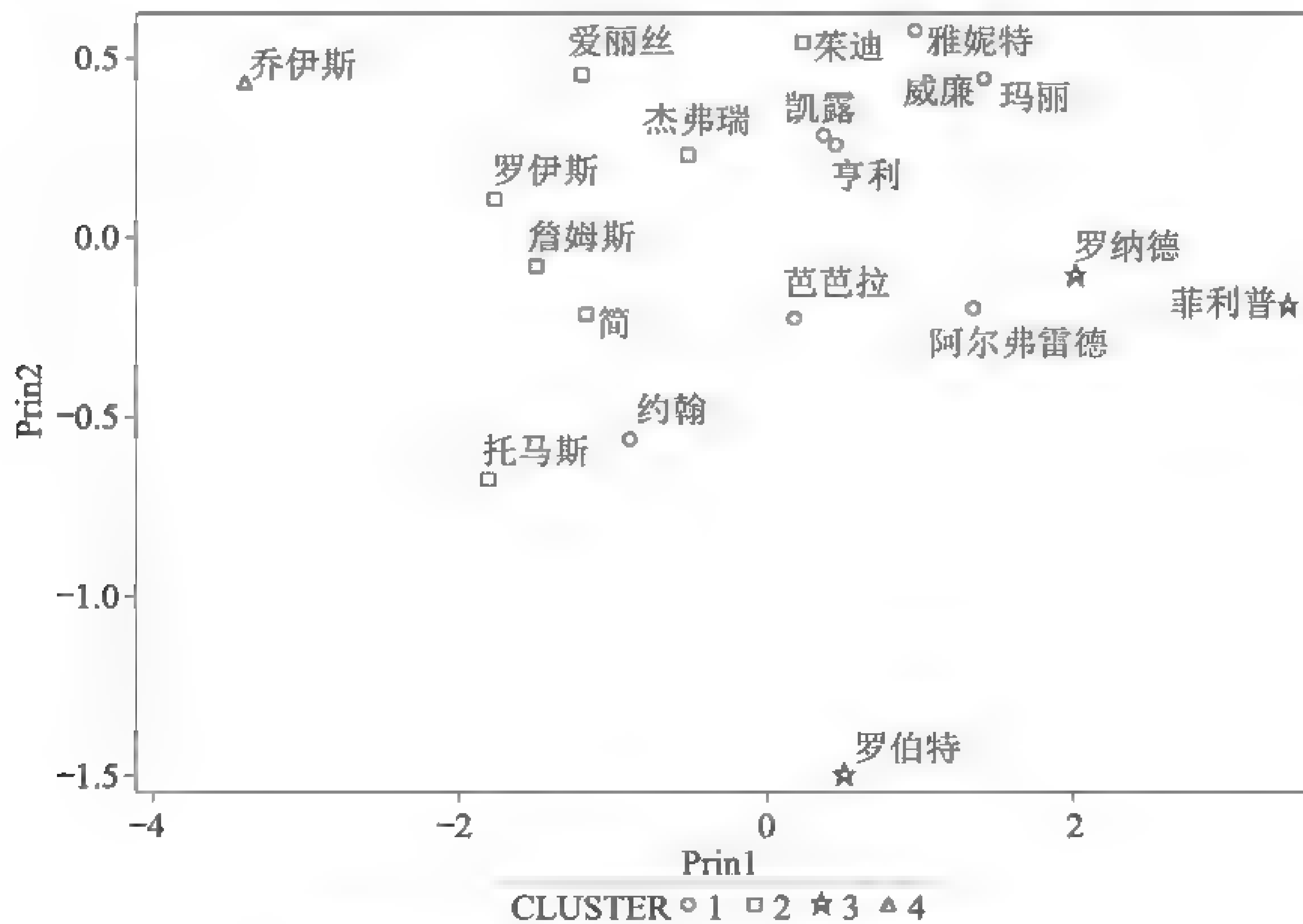


图 25-22　聚类分析结果投影到主分量空间

对聚类结果的投影绘图也可使用 GPLOT 来完成（见程序 25-13），只不过输出的图像比较简略而已。

程序25-13　聚类结果的简单投影

```
symbol1 c=red f=, v='1';
symbol2 c=green f=, v='2';
symbol3 c=blue f=, v='3';
symbol4 c=black f=, v='4';
proc gplot data=class_merged;
  plot prin2*prin1=cluster;
run;
```

聚类分析的最终结果也可以按照类别进行输出，代码如程序 25-14 所示，运行结果如图 25-23 所示。

程序25-14　聚类结果按类别输出

```
proc sort data=class_merged;
  by cluster;
run;
proc print data=class_merged;
  by cluster;
  var name prin1 prin2;
run;
```

SAS 系统

CLUSTER=1

Obs	Name	Prin1	Prin2
1	阿尔弗雷德	1.34442	-0.19965
2	芭芭拉	0.17287	-0.22776
3	亨利	0.45439	0.25852
4	凯露	0.37339	0.28468
5	玛丽	1.41815	0.43911
6	威廉	1.41815	0.43911
7	珍妮特	0.96795	0.57595
8	约翰	-0.89384	-0.56293

CLUSTER=2

Obs	Name	Prin1	Prin2
9	爱丽丝	-1.20046	0.45478
10	简	-1.18162	-0.21387
11	杰弗瑞	-0.50614	0.23057
12	罗伊斯	-1.77682	0.10640
13	托马斯	-1.81080	-0.67325
14	詹姆斯	-1.50895	-0.08255
15	茱迪	0.23001	0.54442

CLUSTER=3

Obs	Name	Prin1	Prin2
16	菲利普	3.39388	-0.19374
17	罗伯特	0.50001	-1.49969
18	罗纳德	2.00851	-0.11012

CLUSTER=4

Obs	Name	Prin1	Prin2
19	乔伊斯	-3.40308	0.43003

图 25-23　聚类结果按类别输出

一般而言，立方聚类准则 CCC 不适用于不规则形状的聚类和最近邻聚类方法，且输入变量之间不能有相关关系，其计算要求协方差矩阵存在特征值。输出图中每个聚类数大于 2 的峰值都是可选方案。

伪 F 统计量则反映当前水平下所有类的分离程度，等于类间距离平方和 B 与类内距离平方和 W 比值相关的统计量，不过它们需要除以各自的伪自由度（$g-1$）和（$n-g$），其中 g 为类别数，n 为观测数。输出图中伪 F 统计量的每个峰值都是可选方案。

伪 t^2 统计量反映最近合并的两个类之间的分离程度，用于 AVERAGE、CENTROID

和 WARD 方法。

最佳聚类数目的确定需要一定的准则，PROC CLUSTER 过程步可输出聚类数准则，包括CCC、伪 F 和伪 t^2 统计量以及R 方等。下面以 SASHELP.IRIS 鸢尾花数据为例说明（见程序 25-15）。

程序25-15 输出聚类数准则

```
proc cluster data=sashelp.iris method=ward outtree=iris tree
  rsquare ccc pseudo ;
  var SepalLength SepalWidth PetalLength PetalWidth ;
run;
```

在输出的聚类数准则图上，CCC 大致在 3 和 5 附近形成波峰，而伪 F 统计量则在 3 附近形成单一波峰，伪 t^2 则在 3 或 6 处有波谷，因此将数据分成 3~6 类是最合适的（见图 25-24）。

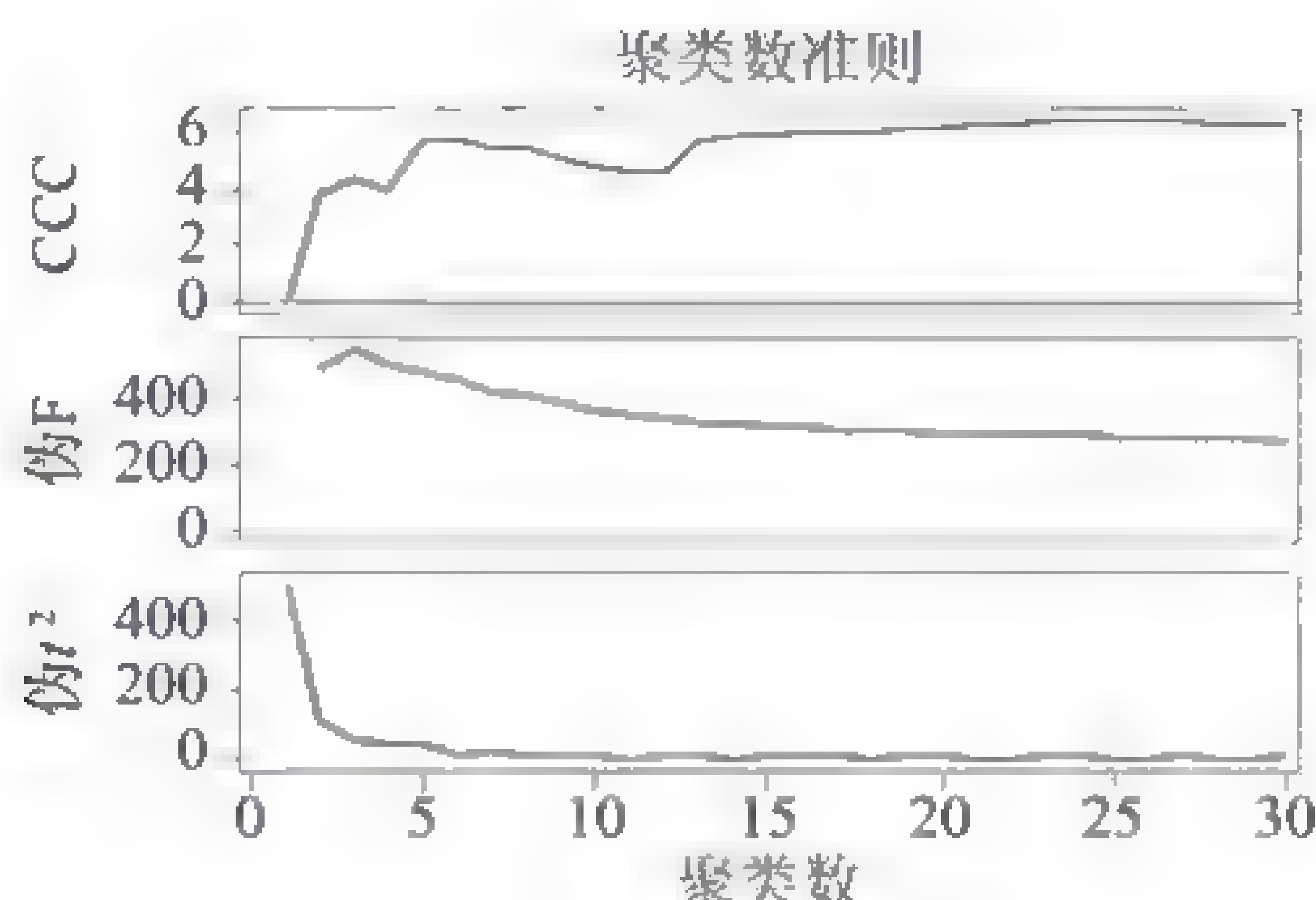


图 25-24 聚类数准则判定

这一点也可从鸢尾花数据的聚类谱系图上直观看到，在作者标注的虚线上进行切割可分别获得 3 和 6 个聚类（见图 25-25）。

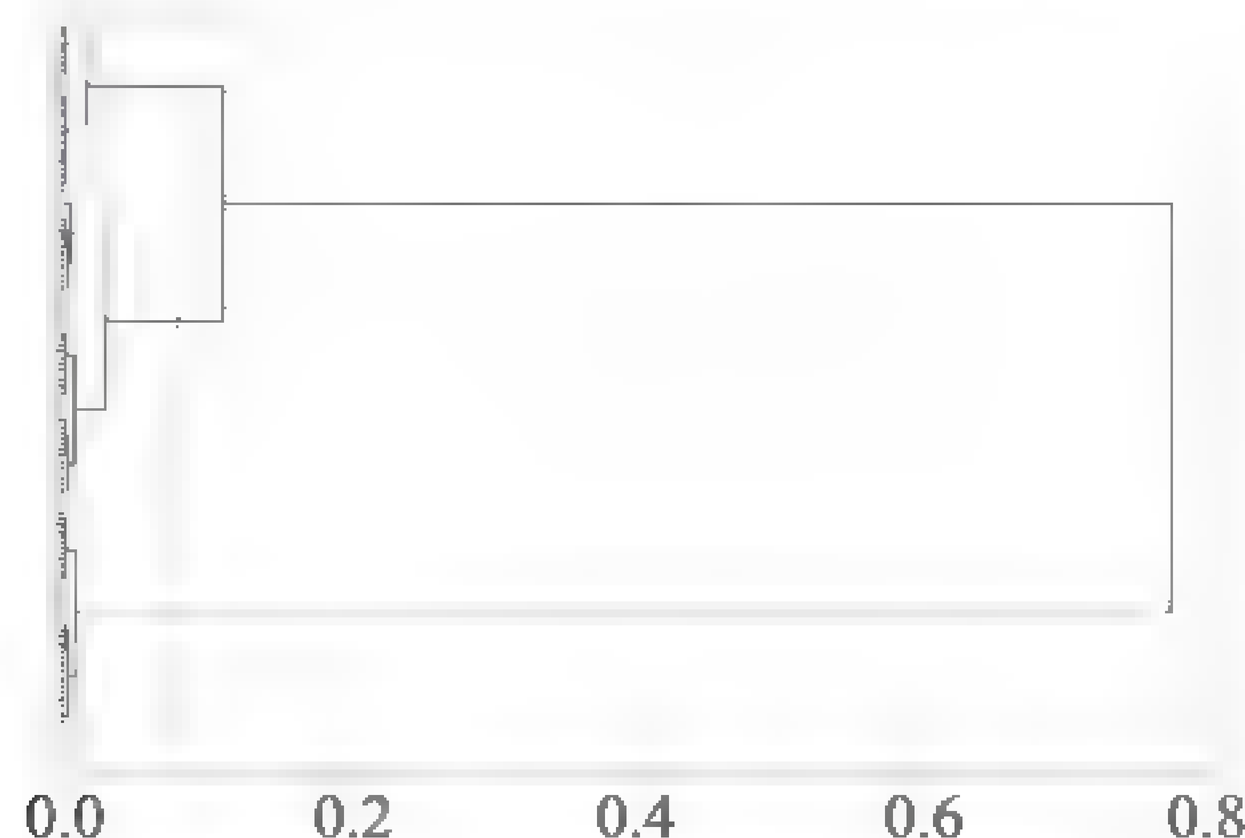


图 25-25 聚类数直观研判

需要特别注意，不同的聚类方法对同样的样本数据可产生不同的分析结果。比如使用 AVERAGE 方法输出，其中立方聚类准则 CCC 可以看出推荐分类数在3，5，6和 23 附近，而伪 F 统计量则指示分成 3，4，6 类为宜，伪 t^2 统计量则指示分类数在 3 或 4 为宜（见程序 25-15B 和图 25-26）。

程序25-15B 不同聚类方法的聚类数准则

```
proc cluster data sashelp.iris method average outtree iris tree
  rsquare ccc pseudo ;
  var SepalLength SepalWidth PetalLength PetalWidth ;
run;
```

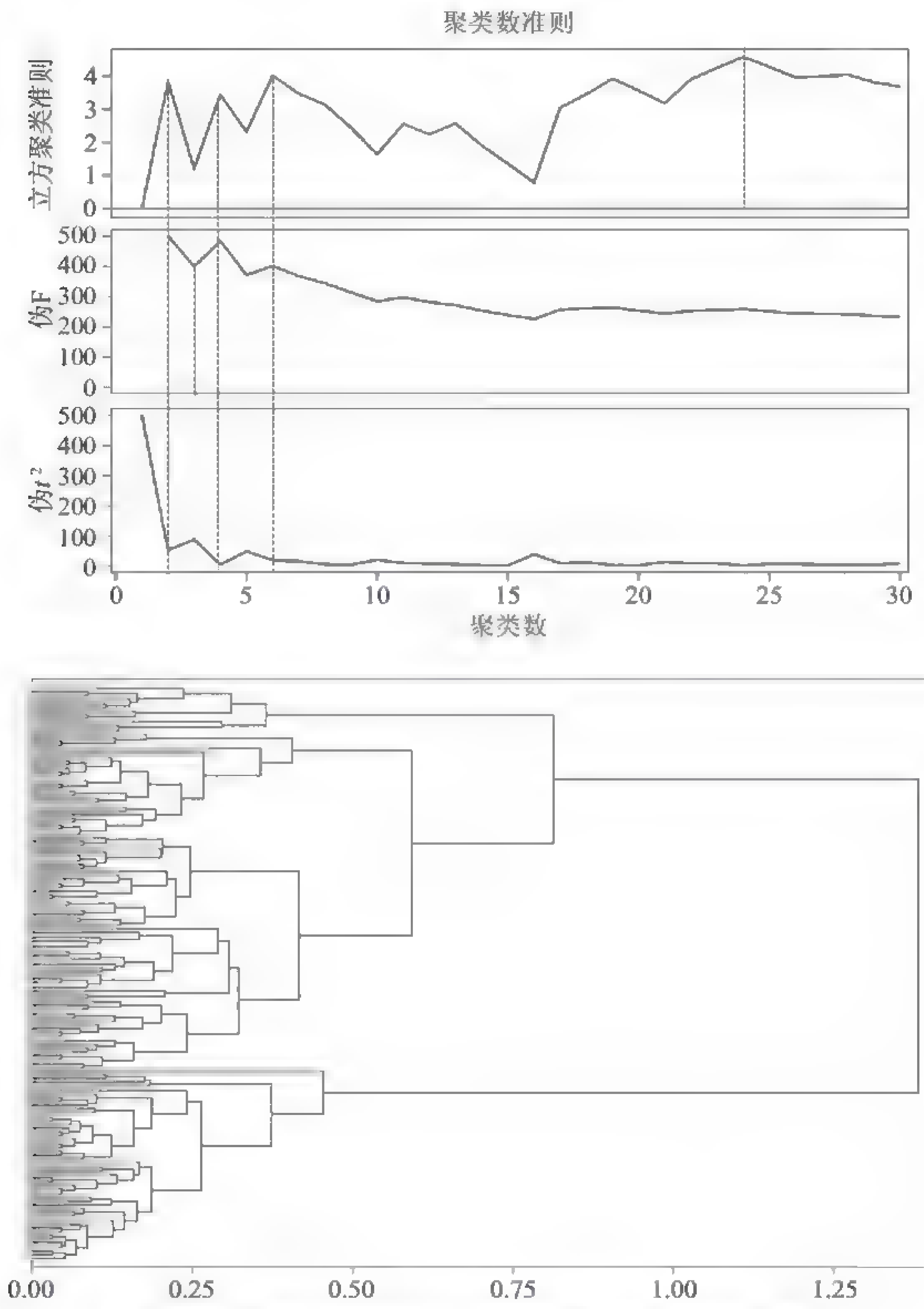


图 25-26 不同聚类方法的聚类数准则（虚线为人为标记线）

25.2.4 R 型聚类分析

大部分聚类是对数据集横向上的观测进行的聚类，称为 Q 型分析。SAS 默认的方法

通常是先求距离尺度后再进行层次聚类（见程序 25-16 及其输出结果图 25-27）：

程序25-16 Q型聚类分析

```
proc distance data=sashelp.class method=dgower out=data_dist;
  var ratio(height  / std=maxabs)
    interval(weight / std=std)
      ordinal(age   / std=std)
    nominal(sex);
  copy name;
run;

proc cluster data=data_dist method=ward plots;
  var dist:;
  id name;
run;
```

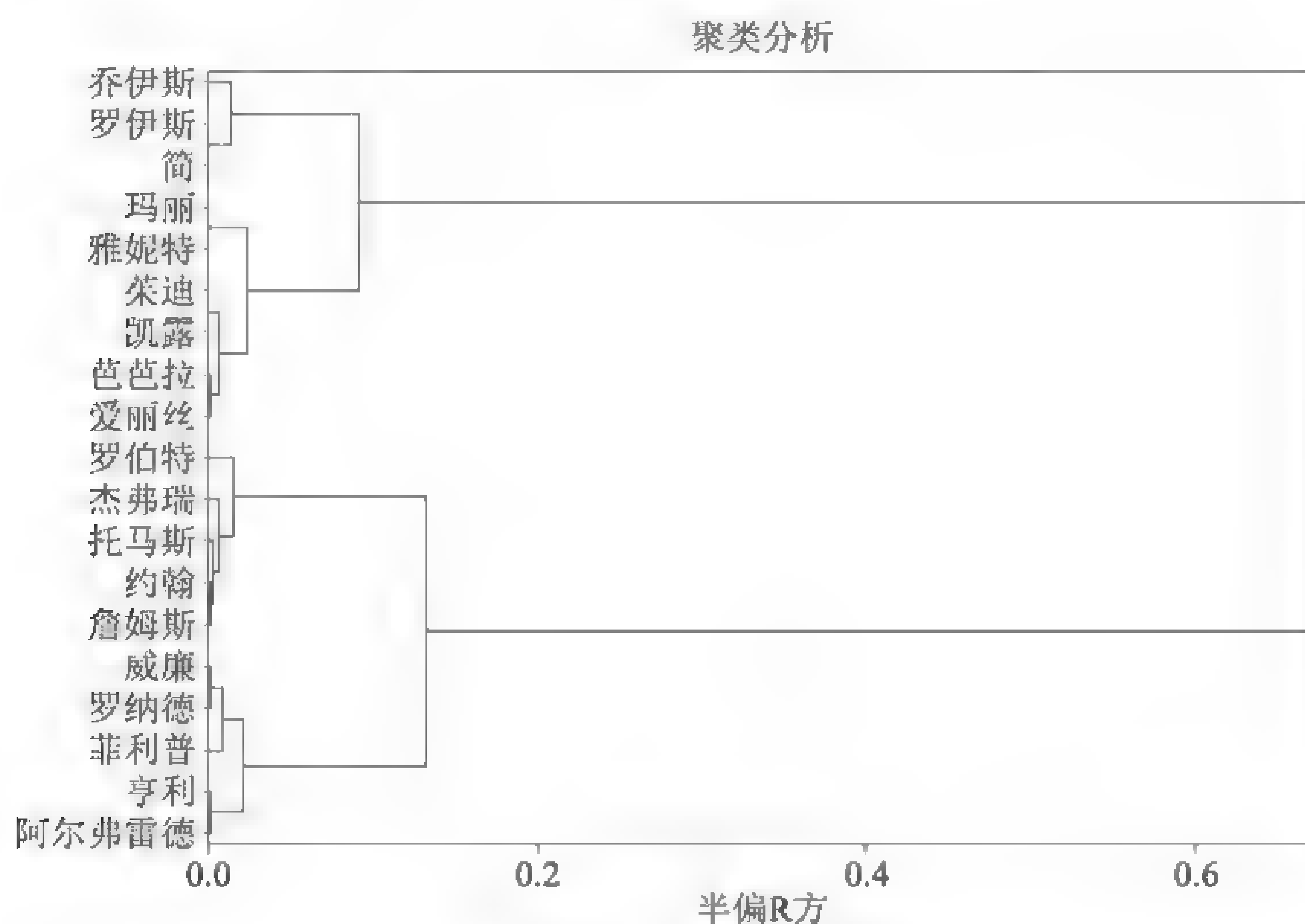


图 25-27 Q 型聚类分析

如果要做 R 型聚类分析，即分析某个数据集纵向上各变量之间的关系，而不是横向上各样本之间的关系。方法之一是先将数据转置然后再对数据执行 Q 型聚类分析即可。比如以前面最简单的 d1 数据集为例执行 R 型聚类（见程序 25-17）。

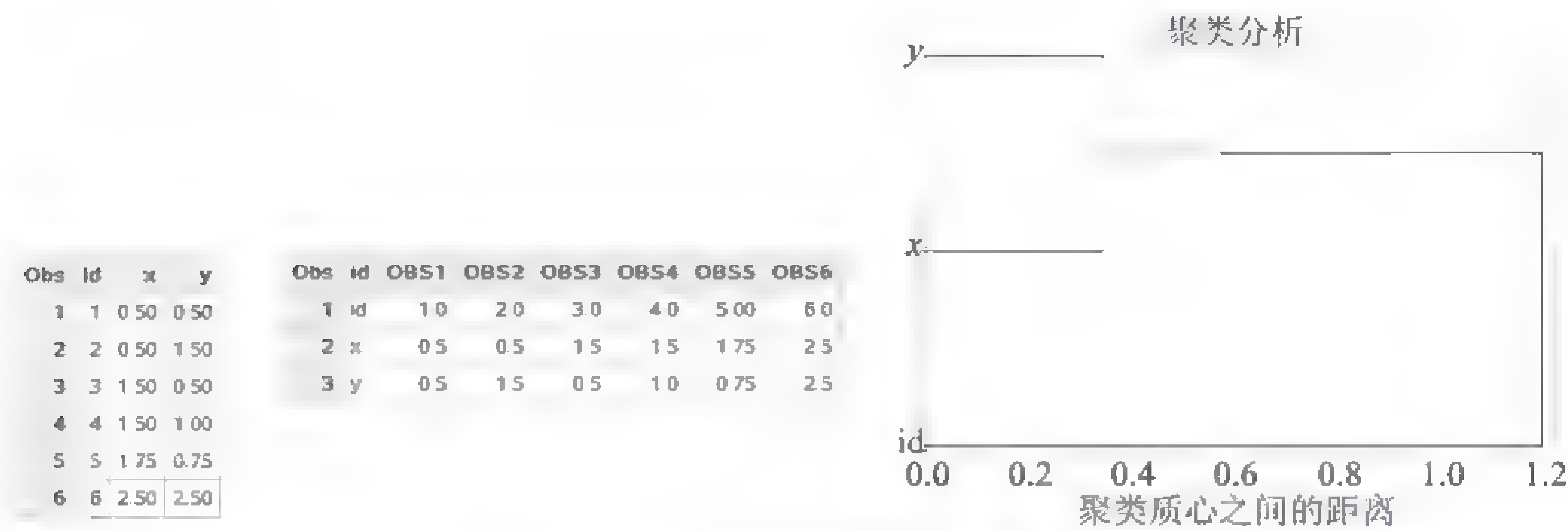
程序25-17 转置数据后执行聚类

```
data d1;
  input id x y @@;
  datalines;
1  0.5  0.5
2  0.5  1.5
3  1.5  0.5
4  1.5  1.0
5  1.75 0.75
6  2.5  2.5
;
proc print data=d1;
run;
```

```
proc transpose data=d1 out=d2 prefix=OBS name=id;
run;
proc print data=d2;
run;

proc cluster data=d2 method=centroid;
  id id;
run;
```

转置前后的数据表和分析结果如图 25-28 所示，聚类显示样本编号 id 明显跟坐标 x, y 不在一个聚类内，大概是因为 x, y 变量用来描述空间坐标，而编号 id 只是样本的索引，它们之间确实没什么关系。



Obs	id	x	y
1	1	0.50	0.50
2	2	0.50	1.50
3	3	1.50	0.50
4	4	1.50	1.00
5	5	1.75	0.75
6	6	2.50	2.50

Obs	id	OBS1	OBS2	OBS3	OBS4	OBS5	OBS6
1	id	1.0	2.0	3.0	4.0	5.00	6.0
2	x	0.5	0.5	1.5	1.5	1.75	2.5
3	y	0.5	1.5	0.5	1.0	0.75	2.5

图 25-28　转置数据的聚类分析

R 型聚类分析的第二种方法是用 SAS 官方提供的专门过程步 PROC VARCLUS 对变量进行聚类。比如可对 SASHELP.CLASS 的 3 个数值变量进行聚类，检查变量 age、height 和 weight 之间的紧密程度（见程序 25-18 及其输出图 25-29）。

程序25-18　对SASHELP.CLASS 数据执行R 型分析

```
proc varclus data=sashelp.class hierarchy plots ;
  var age height weight  ;
run;
```

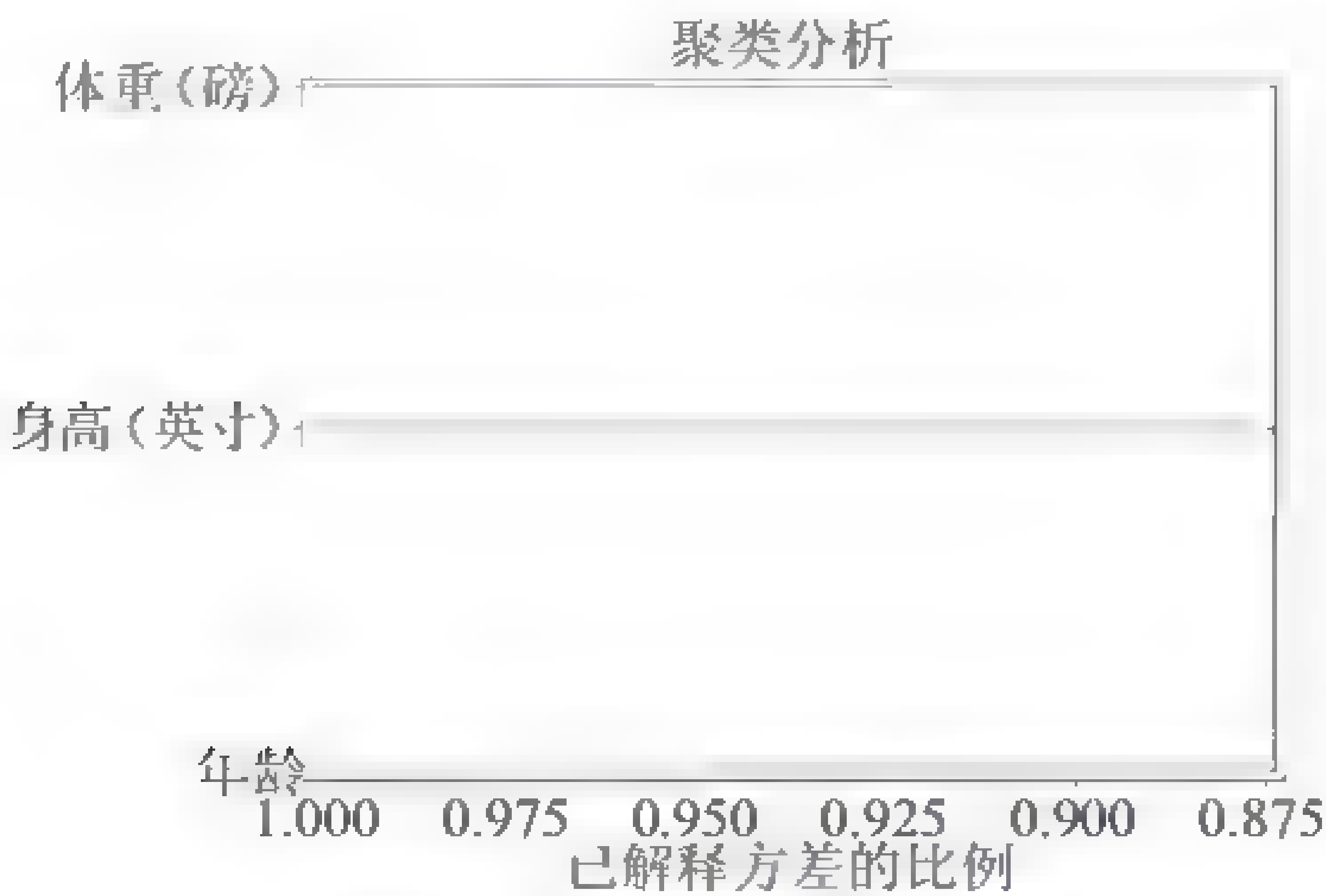


图 25-29　默认 R 型分析结果

由于系统默认采用斜交主成分聚类，而 SASHELP.CLASS 数据的计算中没有满足拆

分准则的聚类，因此输出图 25-29 所示的聚类图，似乎表示 3 个变量之间距离差不多。但我们为了仔细观察变量之间的亲疏关系，可强行指定最大聚类数为 3，运行分析代码（见程序 25-19 及其输出图 25-30），则我们可以看到年龄自成一类，身高体重形成次一级聚类，说明年龄相对而言是更加独立与身高体重的变量，聚类图与我们直觉相符。

程序25-19　对SASHELP.CLASS 数据执行R 型分析，强制指定聚类数为3

```
proc varclus data=sashelp.class hierarchy plots maxc=3 ;
  var age height weight  ;
run;
```

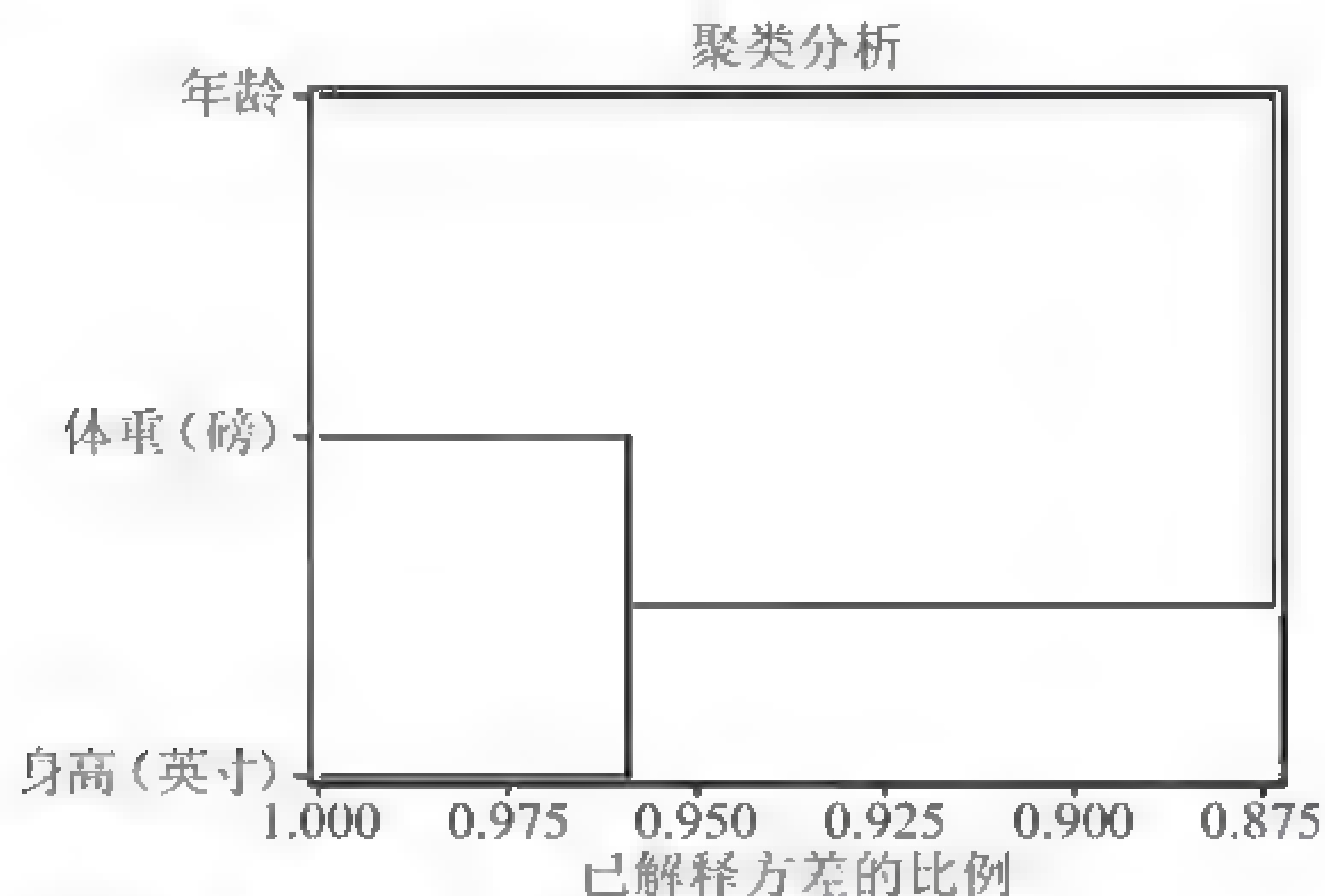


图 25-30　强制聚类数目的 R 型分析

25.3　自己实现聚类算法

在 SAS 中用户可自己编程实现完整的聚类分析算法，下面分别以 K- 均值方法和简单逐步形成分类系统的聚类方法为例加以说明。

25.3.1　K-均值方法

由于 FCMP 函数足以让我们实现任何想要的计算机算法，现在以它来实现 K- 均值方法所涉及的各函数，在函数中所进行的计算都是基于数组实现。为保证完整性和可执行性，对各函数的说明嵌入到如下代码注释中，函数编译后输出到 WORK.FUNCS.KMEANS 中。参照前一节图 25-12 的迭代过程，设计 K- 均值方法的关键数据结构如图 25-31 所示，代码见程序 25-20。

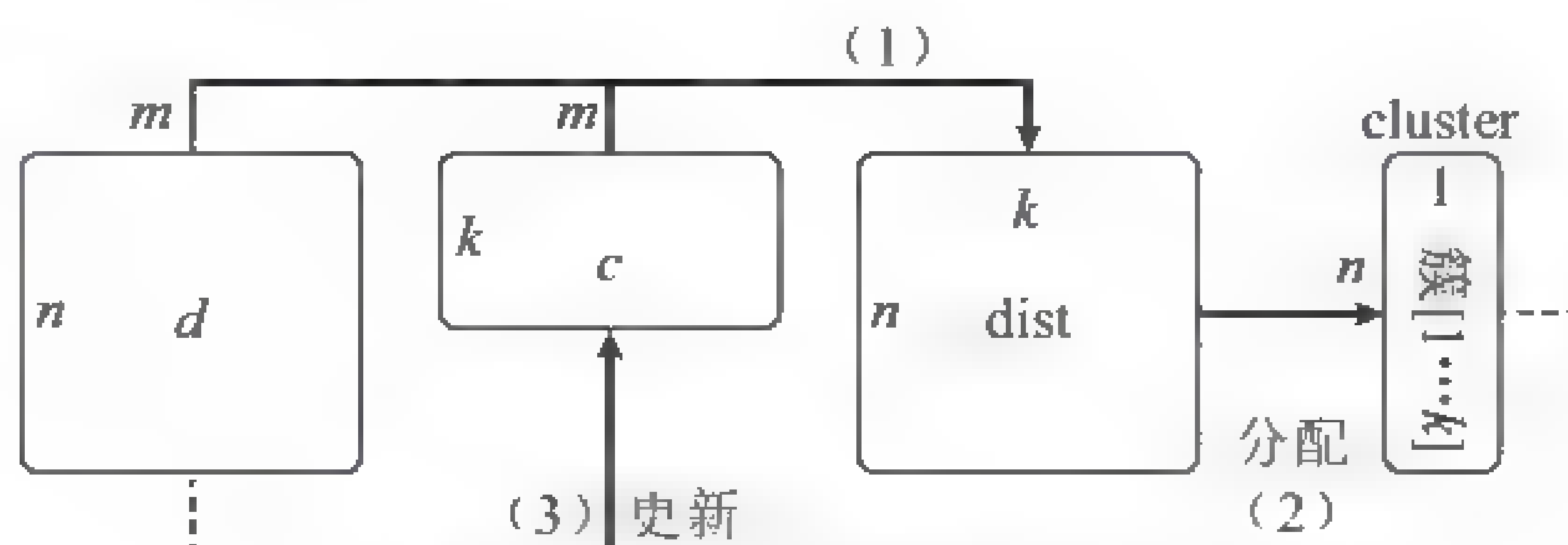


图 25-31　主要数据结构和关系

程序25-20 K-Means 的FCMP函数实现

```
proc fcmp outlib=work.funcs.kmeans;
  /*(1) 计算n个样本 d[n,m] 到 k 个质心 c[k,m] 之间的距离,输出 dist[n,k]*/
  function calcDistanceMatrix(d[*,*],c[*,*], dist[*,*]);
    outarg dist;
    n=dim(d);
    m=dim(d,2);
    k=dim(c);

    if m ^= dim(c,2) then do;
      put 'NOTE: Sample and centroid vector length mismatch';
      return(.);
    end;

    do i=1 to n;
      do j=1 to k;
        sum=0;
        do mm=1 to m;
          sum=sum+ (d[i,mm]- c[j,mm])**2;
        end;
        dist[i,j]= sqrt(sum);
      end;
    end;
    return (1);
  endsub;

  /*(2) 分配阶段：找到每个点最近的质心，将它归入该聚类*/
  function updateClusters(dist[*,*],cluster[*]);
    outarg cluster;
    n=dim(dist);
    k=dim(dist,2);

    if n ^= dim(cluster) then do;
      put 'NOTE: Cluster and distance matrix rows mismatch';
      return(.);
    end;

    do i=1 to n;
      min_cluster=-1;
      min_dist=constant("BIG");
      do j=1 to k;
        if dist[i,j]<min_dist then do;
          min_cluster=j;
          min_dist=dist[i,j];
        end;
      end;
      cluster[i]=min_cluster;/*分配聚类*/
    end;
    return (1);
  endsub;

  /*辅助函数：拷贝数组 s[n] -> t[n]用于保存上一次聚类 */
  function CopyArray(s[*], t[*]);
    outarg t;
    n=dim(s);
    if dim(t)<n then return(.);

    do i=1 to n;
      t[i]=s[i];
    end;
    return (1);
```

```
endsub;

/* (3) 更新阶段：基于n 样本 d[n, m]和聚类 cluster[n] 计算聚类质心 c[k,m]*/
function updateCentroids(d[*,*], cluster[*], c[*,*]);
  outarg c;
  n=dim(d);
  m=dim(d,2);
  k=dim(c);

  if n ^= dim(cluster) then do;
    put 'NOTE: Cluster and distance matrix rows mismatch';
    return(.);
  end;

  array count[1] /nosymbols;
  call dynamic_array(count, k);
  do i=1 to k;
    count[i]=0;
  end;

  do i=1 to k;
    do j=1 to m;
      c[i,j]=0;
    end;
  end;

  do i=1 to n;
    kk=cluster[i];
    count[kk]=count[kk]+1;

    do j=1 to m;
      c[kk, j]=c[kk, j] + d[i,j];
    end;
  end;

  do i=1 to k;
    if count[i]>0 then do;
      do j= 1 to m;
        c[i,j] =c[i,j] / count[i];
      end;
    end;
  end;
  return (1);
endsub;

/*计算已分配类别的组内方差总和：输入样本 d[n,m] 质心 c[k,m] 类别 cluster[n] */
function getTotalDistance(d[*,*],c [*,*],cluster[*]);
  n=dim(d);
  m=dim(d,2);
  k=dim(c);

  if m ^= dim(c,2) then do;
    put 'NOTE: Sample and centroid vector length mismatch';
    return(.);
  end;

  total_distance=0;
  do i=1 to n;
  kk=cluster[i];
  if kk ^= -1 then do;
    sum=0;
    do mm=1 to m;
```

```
          sum=sum+ (d[i,mm] - c[ kk,mm])**2;
        end;
      total distance total distance + sqrt(sum);
    end;
  end;
  return (total distance);
endsub;

/*辅助函数：检测类别分配的成员变化数目，如果为零表示没有变化*/
function getClusterChangeCount(a[*], b[*]);
  n=dim(a);
  if dim(b)^=n then return(.);

  change_count=0;
  do i=1 to n;
    if a[i] ^= b[i] then change_count=change_count+1;
  end;
  return (change_count);
endsub;

/* K-means 主函数:聚类分析的调用入口，基于若干数组
* 输入：样品数据 d[n,m] 聚类数目 k
* 输出：聚类质心 c[k,m] 类别归属 oCluster[n] 距离数据 oDistance [n, m] */
function kmeans(d[*,*], k, c [*,*], oCluster[*], oDistance[*]);
  outarg c;
  outarg oCluster;
  outarg oDistance;

  n=dim(d);

  array dist[1,1] /nosymbols;
  call dynamic_array(dist, n,k);

  array cluster  [1] /nosymbols;
  call dynamic_array(cluster  , n);

  array cluster_orig [1] /nosymbols;
  call dynamic_array(cluster_orig , n);

  rc=calcDistanceMatrix(d, c, dist); /*(1) 计算距离*/
  ret=updateClusters(dist, cluster); /*(2) 分配聚类*/

  rc=CopyArray(cluster, cluster_orig); /*提前保存分类情况*/
  total_dist_orig= constant("BIG");
  iteration=1;
  iterationEnd=0;
  do while(iteration<= 1000 AND iterationEnd=0);

    rc=updateCentroids(d, cluster, c);/*(3) 更新质心*/
    total_dist= getTotalDistance( d,c, cluster); /*评估效用,是否有改进?*/
    if total_dist <total_dist_orig then do;
      rc=CopyArray( cluster, cluster_orig);/*提前保存分类情况*/

      rc=calcDistanceMatrix(d, c, dist);/*(1) 计算距离*/
      rc=updateClusters ( dist, cluster);/*(2) 分配聚类*/

      rc=getClusterChangeCount( cluster, cluster orig);
      if rc>0 then do;
        total dist orig=total dist;
        iteration iteration+1;
      end;
      else do;
```

```
          iterationEnd=1;/*收敛:类别不再变动*/
        end;
      end;
      else do;
        /*终止：没有改进则恢复上一次结果并结束迭代*/
        rc=CopyArray( cluster_orig, cluster);
        rc=updateCentroids( d, cluster, c);
        iterationEnd=1;
      end;
    end;
    /*将结果输出到传入的数组*/
    rc=CopyArray(cluster,oCluster);
    do i=1 to n;
      oDistance[i]=dist[i, cluster[i]];
    end;
    return (1);
  endsub;
run;
quit;
```

程序 25-21 的代码演示了如何调用自己开发的 K-means 聚类算法，出于比较的目的演示程序使用相同的演示数据集 d1。

程序25-21 自定义K-Means 算法实现演示程序

```
data d1;
    input id x y @@;
    datalines;
1  0.5  0.5
2  0.5  1.5
3  1.5  0.5
4  1.5  1.0
5  1.75 0.75
6  2.5  2.5
;

options cmplib=work.funcs; /*引用 WORK.FUNCS 函数库*/
data _null_;
  /* (1) 读入外部数据集 d1 到二维数组 dat[6,2] 中*/
  array dat[6,2] _temporary_;
  array row[2] x y ;
  put 'Data';
  do i=1 to rows;
    set d1 point=i nobs=rows;
    do j=1 to dim(dat,2);
    dat[i,j]=row[j];
      put dat[i,j] 5.2 @;
    end;
    put;
  end;

  /* (2) 设置 k=3 且指定前 3 个数据点为初始质心 */
  k=3;
  put 'Init Centeroid (k=' k ")";
  array c[3,2] _temporary_;
  do i=1 to k;
    do j= 1 to dim(c,2);
      c[i,j]=dat[i,j];
      put c[i,j] 5.2 @;
    end;
    put;
```

```
  end;

  /* (3) 准备数组接收结果 */
  array ocluster[6]  temporary ;
  array odistance[6]  temporary ;
  rc=kmeans(dat, k, c , ocluster,odistance);

  put "X     Y     CLUSTER DISTANCE";
  do i=1 to dim(dat);
    put dat[i,1] 4.2 " " dat[i,2] 4.2 " " ocluster[i] 7.0 " "
      odistance[i] 5.3 ;
  end;
  stop;
run;
```

系统输出如下：

```
Data
 0.50 0.50
 0.50 1.50
 1.50 0.50
 1.50 1.00
 1.75 0.75
 2.50 2.50
Init Centeroid (k=3 )
 0.50 0.50
 0.50 1.50
 1.50 0.50
X    Y    CLUSTER DISTANCE
0.50 0.50       1 0.500
0.50 1.50       1 0.500
1.50 0.50       3 0.264
1.50 1.00       3 0.264
1.75 0.75       3 0.167
2.50 2.50       2 0.000
```

如果要使用SAS的图形过程步对自定义K-Means算法输出进行绘图，只需要将前面的程序25-21改动两处即可（见程序25-22斜体部分），其中之一是指定输出数据集名称，另一处插入7行代码即可。

程序25-22　将K-means 结果输出到数据集

```
data d1_cls; /* 修改：指定输出数据集名称 */
  /*…此处省略准备数组和调用 kmeans 函数代码，参考前面代码*/

  put "X     Y     CLUSTER DISTANCE";
  do i=1 to dim(dat);
    put dat[i,1] 4.2 " " dat[i,2] 4.2 " " ocluster[i] 7.0 " " odistance[i] 7.5 ;
    /*修改起始：指定要输出的变量名称*/
    cluster=ocluster[i];
    distance=odistance[i];
    do j=1 to 2;
      row[j]=dat[i,j];
    end;
    output;
    keep x y cluster distance;
    /*修改结束*/
  end;
  stop;
run;
```

此时就可看到输出的数据集 D1_CLS 如下，然后用 SGPLOT 过程步按照聚类分组输出（见程序 25-23），其中通过指定 MARKERCHAR=CLUSTER 将在输出图上直接输出聚类编号。

程序25-23　对聚类输出数据集d1_cls 绘图

```
proc sgplot data=d1_cls; /*调用 SGPLOT 绘图与 proc fastclus 输出一样*/
  scatter y=y x=x / group=Cluster markerchar=Cluster;
  xaxis min=0 max=3 grid;
  yaxis min=0 max=3 grid;
run;
```

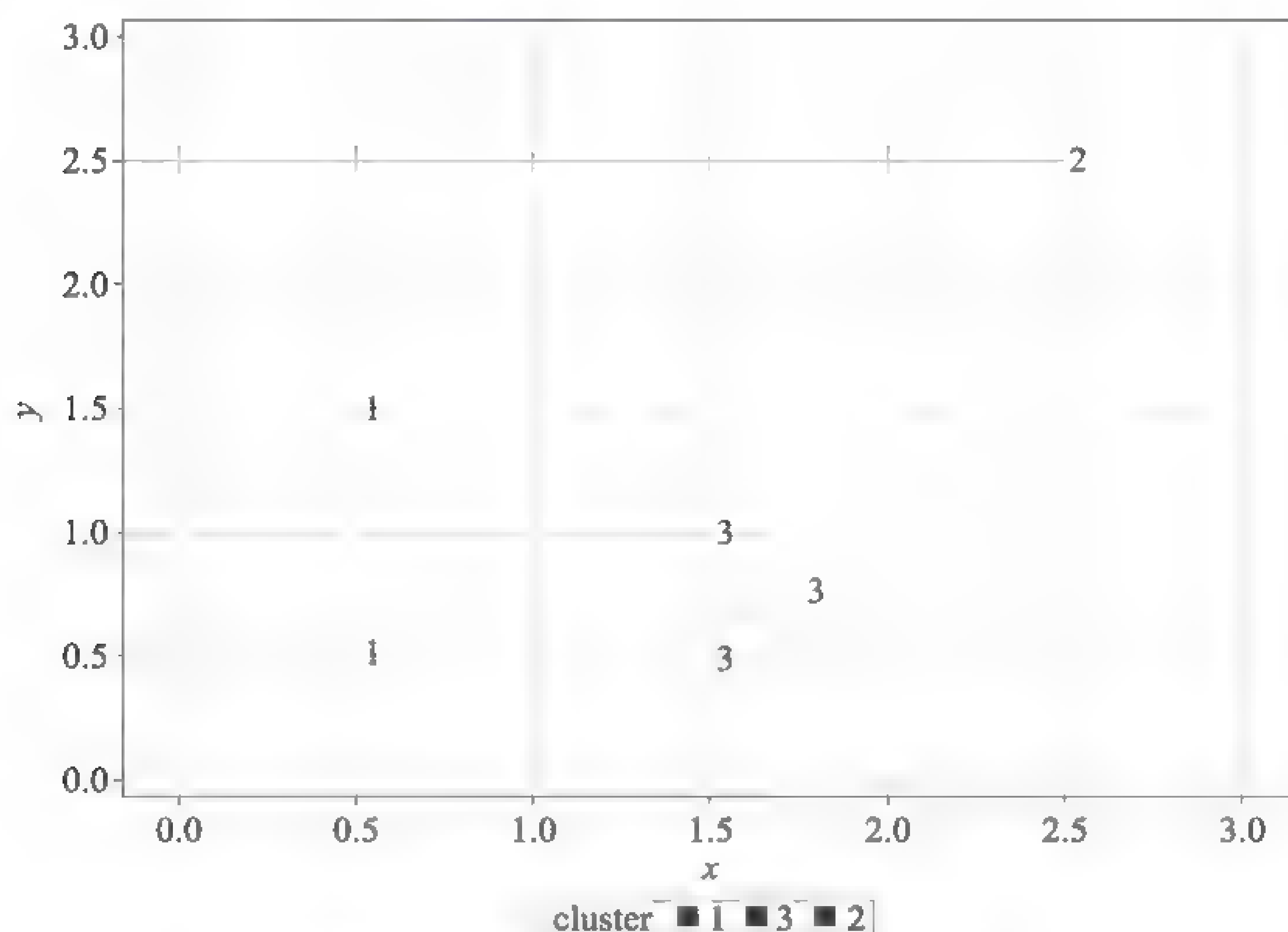


图 25-32　自定义 K-Means 算法结果绘图

实际上，可对比自己实现的 K- 均值方法跟 SAS 官方的 K- 均值方法的结果（见程序 25-24 及其输出图 25-33），发现它们计算结果完全一样。左侧为自己实现的算法，右侧为 PROC FASTCLUS 结果。

程序25-24　验证 K-均值算法实现的结果

```
proc print data=d1_cls; /*打印自定义K-均值输出的聚类结果*/
run;

proc fastclus data=d1 maxclusters=3 out=d1_cls2;
  id id;
run;
proc print data=d1_cls2;run; /*打印 SAS 官方的 K- 均值实现输出结果*/
```

Obs	x	y	cluster	distance
1	0.50	0.50	1	0.50000
2	0.50	1.50	1	0.50000
3	1.50	0.50	3	0.26352
4	1.50	1.00	3	0.26352
5	1.75	0.75	3	0.16667
6	2.50	2.50	2	0.00000

Obs	id	x	y	CLUSTER	DISTANCE
1	1	0.50	0.50	2	0.50000
2	2	0.50	1.50	2	0.50000
3	3	1.50	0.50	3	0.26352
4	4	1.50	1.00	3	0.26352
5	5	1.75	0.75	3	0.16667
6	6	2.50	2.50	1	0.00000

图 25-33 对比输出的聚类和距离

25.3.2 逐步形成分类系统

程序 25-25 为用户自定义实现逐步聚类分析方法的完整代码，其实现思路完全是从程序员角度出发。主要包括 3 部分。

（1）数据标准化 Min-Max 标准化方法函数 MaxMinToOne 及返回特定列极值的函数 GetColExtreme。

（2）计算欧氏距离矩阵函数 EuclideanDistance。

（3）逐步形成聚类分析主函数 StepWiseQ 以及上三角矩阵搜索函数 FindMinMaxPos。注意其中有个标志数组 w 用于控制逐步凝聚的过程。

程序25-25 自定义逐步形成分类系统方法

```
proc fcmp outlib=work.funcs.stepwise library=funcs;
  /*1) 实现数据标准化：Min-Max 标准化方法和辅助函数 GetComExtreme*/
  function GetColExtreme(d[*,*],c, bMax);
    cols=dim(d,2);
    if c <= cols then do;
      rows=dim(d,1);
      colextreme=d[1,c];
      do i=2 to rows;
        if (bMax=0 AND d[i,c]<colextreme) or (bMax=1 AND d[i,c]>colmax) then
          colextreme=d[i,c];
      end;
      return( colextreme);
    end;
    return (.);
  endsub;

  function MaxMinToOne(d[*,*],o[*,*]);
    outargs o;
    rows=dim(d,1);
    cols=dim(d,2);

    if rows^=dim(o,1) or cols^=dim(o,2) then return(0);
    array colmax[1] /nosymbols;
    call dynamic array(colmax, cols);
    array colmin[1] /nosymbols;
    call dynamic array(colmin, cols);
    do c=1 to cols;
      colmax[c]=GetColExtreme(d, c, 1);
```

```
    colmin[c]=GetColExtreme(d, c, 0);
    if colmax[c]-colmin[c]=0 then return(0);
  end;
  do c=1 to cols;
    do r=1 to rows;
      o[r,c]=(d[r,c]-colmin[c])/(colmax[c]-colmin[c]);
    end;
  end;
  return (1);
endsub;

/*2) 计算欧氏距离矩阵：对角线为 0 ,上下三角对称且[0,1] 越小表示样品间越亲密*/
function EuclideanDistance(d[*,*],o[*,*]);
  outarg o;
  n=dim(d,1);
  m=dim(d,2);
  if m=0 then return(0);

  o_rows=dim(o,1); o_cols=dim(o,2);
  if o_rows^=o_cols or o_rows^=n then return(0);

  do r1=1 to n;
    o[r1, r1]=0;
    do r2=r1+1 to n;
      sum=0;
      do c=1 to m;
        diff=d[r1,c] - d[r2,c];
        sum=sum+ diff * diff;
      end;
      o[r1, r2]=sqrt( sum  );
      o[r2, r1]=o[r1, r2];
    end;
    end;
  return (1);
endsub;

/*3) 逐步形成聚类分析主函数 StepWiseQ 以及搜索上三角矩阵中索引值不为零的列中的最大/最小
     值位置 FindMinMaxPos*/
function FindMinMaxPos(d[*,*], pflag[*], minmaxrow, minmaxcol, bMax);
  outarg minmaxrow; outarg minmaxcol;

  rows=dim(d,1);
  cols=dim(d,2);

  if dim(pflag)^=rows then return(0);
  flag=0;
  do row=1 to rows until (flag=1);
    do col=row+1 to cols until (flag=1);
      if pflag[col]^=0 then do;
        minmaxrow=row;
        minmaxcol=col;
        minmaxdat=d[row, col];
        flag=1;
      end;
    end;
  end;
  if flag=0 then do;
    minmaxrow=-1;
    minmaxcol=-1;
    return(0);/*返回 -1, -1 表示没有答案*/
  end;
```

```
  do row=1 to rows ;
    do col=row+1 to cols  ;
      if pflag[col]^=0 then do;/*忽略掉已选中的因子*/
        if (minmaxdat> d[row, col] and bMax=0) or
           (minmaxdat< d[row, col] and bMax=1) then do;
          minmaxdat= d[row, col];
          minmaxrow=row;
          minmaxcol=col;
        end;
      end;
    end;
  end;
  return(1);
endsub;

function StepWiseQ( d[*,*], pi[*], pj[*], pv[*]);
  outarg pi ; outarg pj; outarg pv;
  rows=dim(d,1);
  cols=dim(d,2);

  array dat[1,1] /nosymbols;
  call dynamic_array(dat, rows, cols);
  do r=1 to rows;
    do c=1 to cols;
      dat[r,c]=d[r,c];
    end;
  end;

  ret=MaxMinToOne(dat,dat); /*数据标准化*/
  if ret=0 then return(0);
  array buf[1,1] /nosymbols;/*为逐步分析开辟中间缓冲区*/
  call dynamic_array(buf, rows, rows);

  /*初始化标志数组*/
  array w[1] /nosymbols;
  call dynamic_array(w, rows);
  do r=1 to rows;
    w[r]=1;
  end;

  ret=EuclideanDistance(dat,buf);

  do index=1 to rows-1;
    /*根据分类尺度,找出关系最近的一对 maxi, maxj */
    ret=FindMinMaxPos(buf, w, maxi, maxj, 0);
    if ret=0 then return (0);
    maxv=buf[ maxi, maxj];

    /*得到最小的元素 [i][j]归类*/
    pi[index]=maxi;
    pj[index]=maxj;
    pv[index]=maxv;

    /*将两行数据加权平均*/
    do c=1 to cols;
      dat[maxi, c] =(dat[maxi,c] * w[ maxi] + dat[maxj, c] * w[maxj] )
        / (w[maxi] + w[maxj]);
    end;
    w[maxi] = w[maxi] + w[maxj];
    w[maxj] =0;
```

```
      ret MaxMinToOne(dat, dat); /*数据重新预处理*/
      if ret=0 then return(0); /*如果不能成功执行分析运算*/
      /*重新计算数据*/
      ret=EuclideanDistance(dat,buf);
    end;
    return(1);
  endsub;
run;
quit;
```

而程序25-26则演示了如何调用自己开发的逐步形成聚类分析算法：

程序25-26　自定义逐步形成分类系统演示代码

```
options cmplib=work.funcs;
data d1;
  input x y @@;
  datalines;
0.5 0.5
0.5 1.5
1.5 0.5
1.5 1.0
1.75 0.75
2.5 2.5
;

data _null_;
  array dat[6,2] _temporary_;
  array col[2]  x y ;
  do i=1 to n;
    set d1 point=i nobs=n;
    do j=1 to dim(dat,2);
      dat[i,j]=col[j];
    end;
  end;

  array pi[5] _temporary_;
  array pj[5] _temporary_;
  array pv[5] _temporary_;

  ret=StepWiseQ(dat, pi, pj, pv);

  put "谱系数据";
  do k=1 to dim(pv);
    put pi[k] "," pj[k] "=" pv[k] 5.3;
  end;
  stop;
run;
```

运行上面的代码将在日志中生成如下聚类谱系数据，它能够反映各观测之间的聚合顺序及其聚类时的欧氏距离值：

```
谱系数据
3 ,5 =0.177
3 ,4 =0.198
1 ,2 =0.500
1 ,3 =0.560
4 ,6 =0.992
```

程序25-26只是输出简单的文本信息，为了能在SAS中绘制聚类谱系图，需要修改该程序，将结果输出到外部数据集D1_TREE中，然后就可调用PROC TREE来绘制聚类谱系图。其完整代码如程序25-27所示。

程序25-27 生成 PROCTREE结构良好数据

```
data d1_tree; /*指定输出数据集名称*/
  array dat[6,2] _temporary_;
  array col[2] x y ;
  do i=1 to n;
    set d1 point=i nobs=n;
    do j=1 to dim(dat,2);
      dat[i,j]=col[j];
    end;
  end;

  array pi[5] _temporary_;
  array pj[5] _temporary_;
  array pv[5] _temporary_;

  ret=StepWiseQ(dat, pi, pj, pv);

  put "谱系数据";
  do k=1 to dim(pv);
    put pi[k] "," pj[k] "=" pv[k] 5.3;
  end;

  /*修改开始：扩展如下代码来生成 PROC TREE 所需数据结构*/
  length _name_ $ 12;
  length _parent_ $ 12;
  label _name_= "观测或聚类的名称"
      _parent_= "观测或聚类的父级"
      _height_="聚类质心之间的距离";

  array cluster[6] $ _temporary_;
  array height[5] $ _temporary_;
 _lastheight_=0;

  do index=1 to dim(pv);
    _parent_id_=  dim(pv)-index+1 ; /*为与 SAS 兼容采用逆向编号 5..1*/
    _parent_= trim(left("CL")) || trim(left(_parent_id_));

    i=pi[index];
    j=pj[index];

    height[ _parent_id_ ]= pv[index];

    if cluster[ i]=" " then do;
      if cluster[ j ]=" " then do;
        _name_= i;
        _height_ = 0;
        cluster[ i ]=_parent_id_;
        output;

        _name_ = j;
        cluster[ j ]  =_parent_id_;
        _height_ = 0;
        output;
      end;
      else do;
```

```
    clusterid=cluster[j];
    do k=1 to dim(cluster);
      if cluster[k]=clusterid then do;
        cluster[k]=_parent_id_;
      end;
    end;

    _name_=i;
    cluster[ i ]=_parent_id_;
    _height_=height[ clusterid ];
    output;

    _name_= trim(left("CL")) || trim(left(clusterid));
    cluster[ j ]=_parent_id_;
    _height_= 0;
    output;
  end;
end;
else do;
  if cluster[ j ]=" " then do;
    clusterid=cluster[i];
    do k=1 to dim(cluster);
      if cluster[k]=clusterid then do;
        cluster[k]=_parent_id_;
      end;
    end;

    _name_= trim(left("CL")) || trim(left(clusterid));
    cluster[ i ]=_parent_id_;
    _height_= height[ clusterid ];
    output;

    _name_= j;
    cluster[ j ]=_parent_id_;
    _height_= 0;
    output;
  end;
  else do;
    clusterid=cluster[i];
    do k=1 to dim(cluster);
      if cluster[k]=clusterid then do;
        cluster[k]=_parent_id_;
      end;
    end;

    _name_= trim(left("CL")) || trim(left(clusterid));
    cluster[ i ]=_parent_id_;
    _height_= height[ clusterid ];
    output;

    clusterid=cluster[j];
    do k=1 to dim(cluster);
      if cluster[k]=clusterid then do;
        cluster[k]=_parent_id_;
      end;
    end;

    _name_= trim(left("CL")) || trim(left(clusterid));
    cluster[ j ]=_parent_id_;
    _height_= height[ clusterid ];
    output;
```

```
      end;
    end;
     lastheight =pv[index];
  end;

   name = trim(left("CL")) || trim(left(l));
   parent =" ";
   height = _lastheight_;
  output;

  keep  name   parent   height ;
  /*修改结束*/
  stop;
run;
proc tree data=dl_tree;
run;
```

运行该代码调用 PROC TREE 后输出的聚类结果如图 25-34 所示。

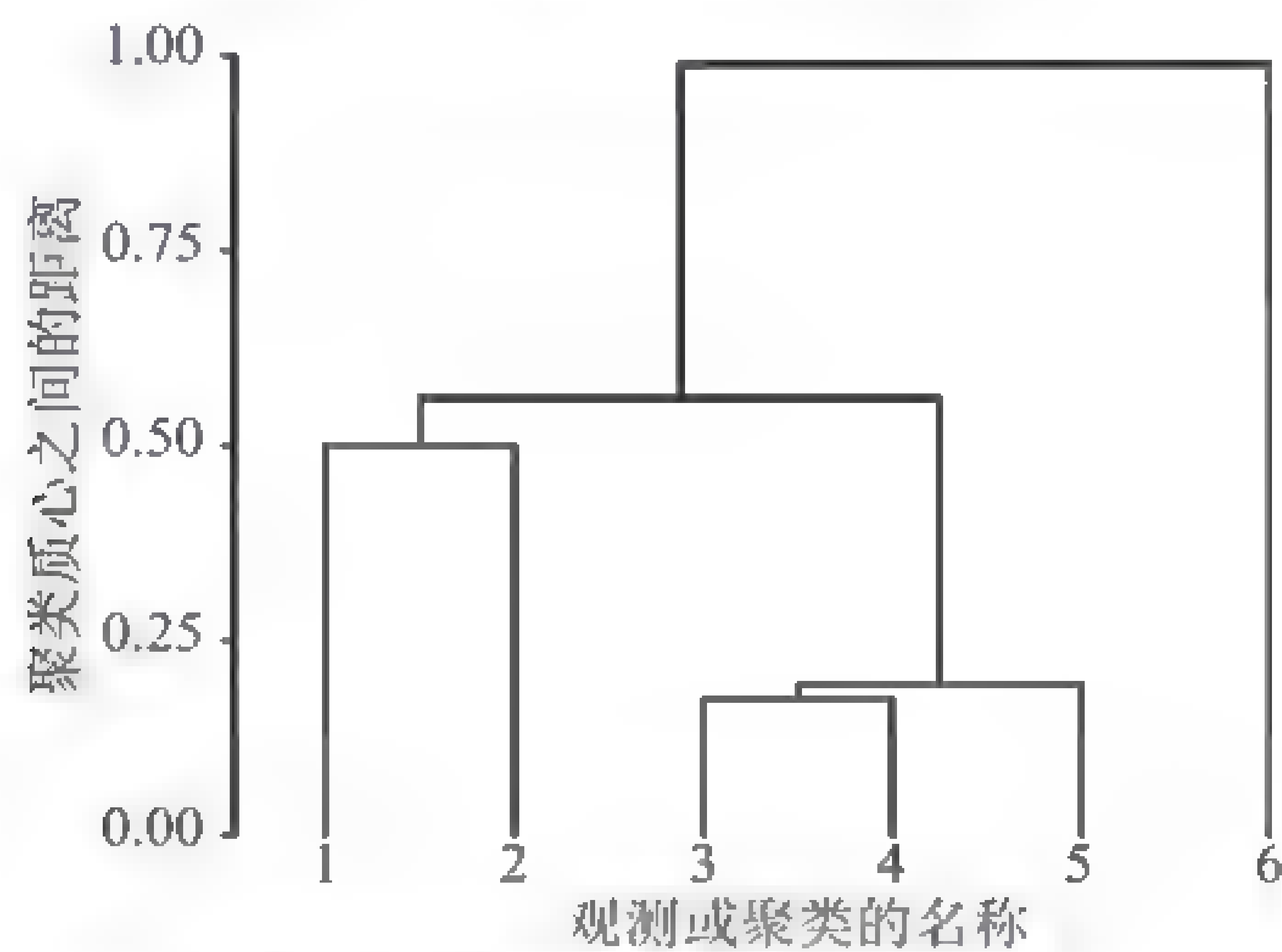


图 25-34　聚类谱系图（默认）

与前面自定义实现的逐步聚类方法等价的 SAS 程序如下所示（见程序 25-29），输出具有完全相同的拓扑结构，但距离的具体计算上稍有不同（见图 25-35 和图 25-36）。

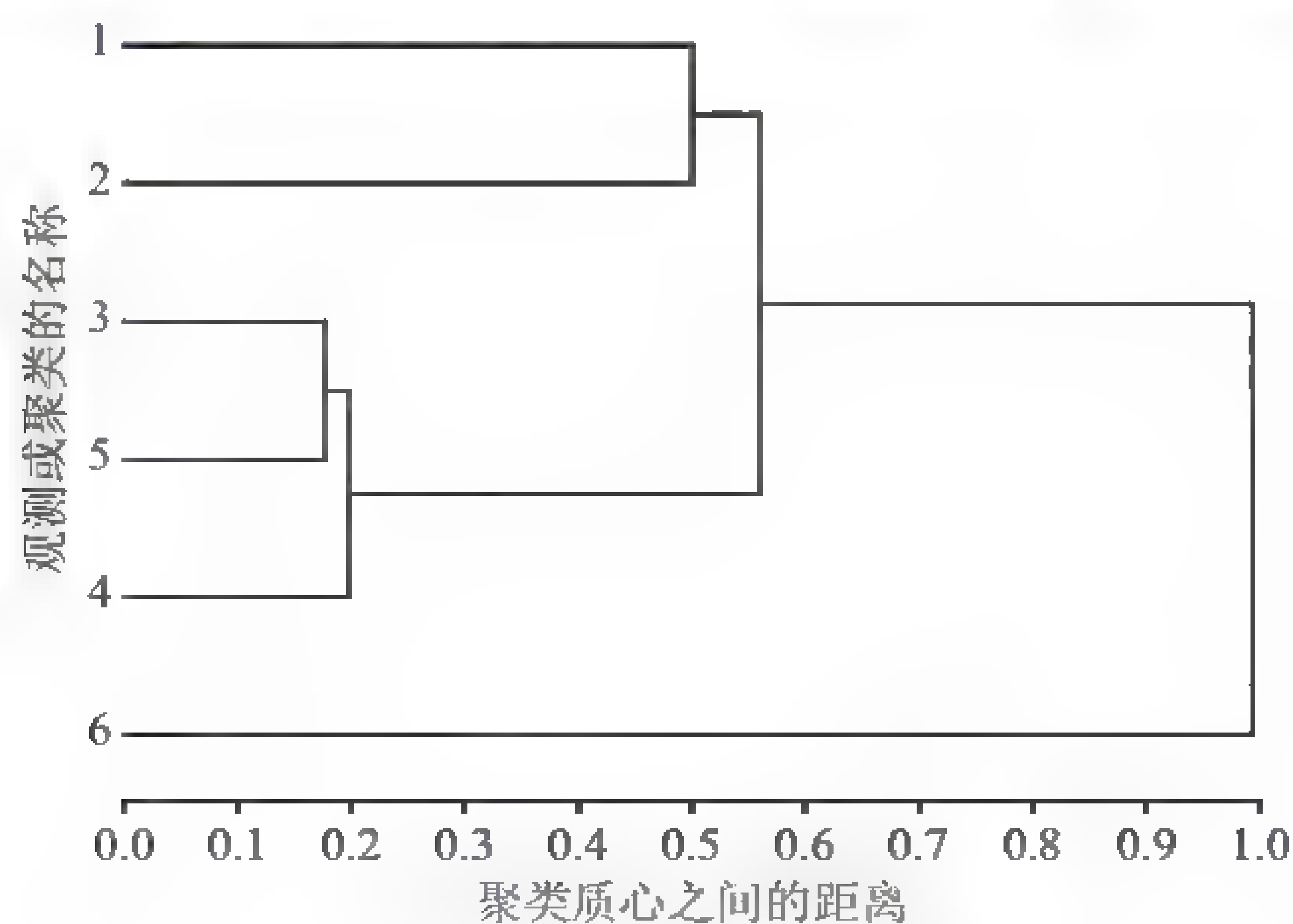


图 25-35　聚类谱系图（横向）

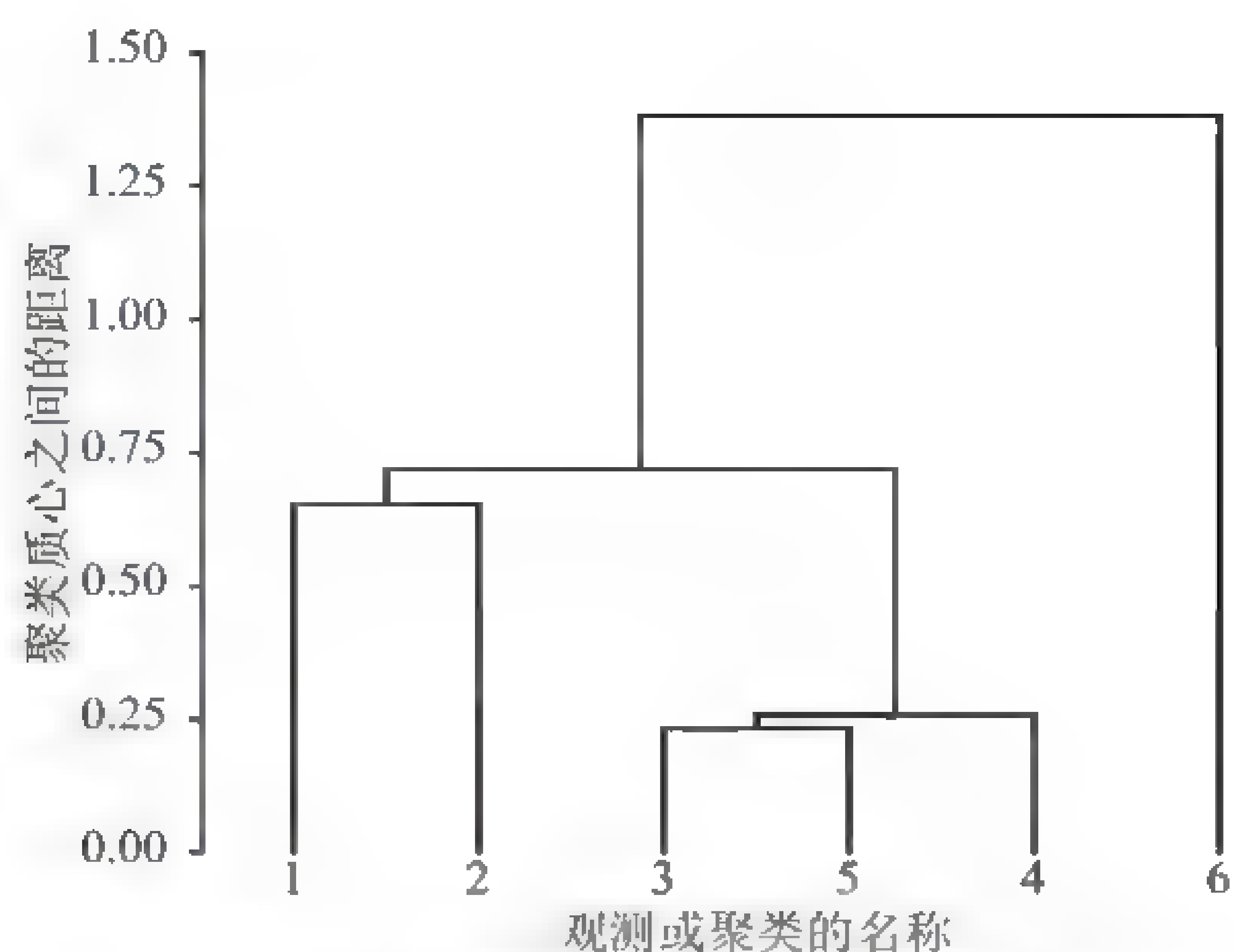


图 25-36　逐步层次聚类输出的谱系图

若要生成坐标轴为横向的聚类谱系图，可在 PROC TREE 上添加 HORIZONTAL 选项（见程序 25-28）。

程序25-28　生成横向聚类谱系图

```
proc tree data=d1_tree horizontal;
run;
```

程序25-29　数据科学家版本的逐步层次聚类

```
data d1;
  input id x y @@;
  datalines;
1 0.5 0.5
2 0.5 1.5
3 1.5 0.5
4 1.5 1.0
5 1.75 0.75
6 2.5 2.5
;

proc cluster data=d1 method=centroid outtree=d1_tree;
  id id;
run;

data d1_tree;
  set d1_tree;
  keep _name_ _parent_ _height_;
run;
proc tree data=d1_tree  ;
run;
```

附录：聚类度量的自定义实现

```
proc fcmp outlib=work.funcs.measure library=funcs;
  /* 距离1：曼哈顿距离 */
  function ManhattanDistance(d[*,*],o[*,*]);
    outarg o;
    n=dim(d,1); m=dim(d,2);
    if m=0 then return(0);
    o_rows=dim(o,1); o_cols=dim(o,2);
    if o_rows^=o_cols or o_rows^=n then return(0);

    do r1=1 to n;
      o[r1, r1]=0;
      do r2=r1+1 to n;
        sum=0;
        do c=1 to m;
          diff=d[r1,c] - d[r2,c];
          sum=sum+ abs(diff);
        end;
        o[r1, r2]= sum;
        o[r2, r1]=o[r1, r2];
      end;
    end;
    return (1);
  endsub;

  /* 距离2：欧氏距离 */
  function EuclideanDistance(d[*,*],o[*,*]);
    outarg o;
    n=dim(d,1);m=dim(d,2);
    if m=0 then return(0);
    o_rows=dim(o,1); o_cols=dim(o,2);
    if o_rows^=o_cols or o_rows^=n then return(0);

    do r1=1 to n;
      o[r1, r1]=0;
      do r2=r1+1 to n;
        sum=0;
        do c=1 to m;
          diff=d[r1,c] - d[r2,c];
          sum=sum+ diff * diff;
        end;
        o[r1, r2]=sqrt( sum  );
        o[r2, r1]=o[r1, r2];
      end;
    end;
    return (1);
  endsub;

  /*距离3：切比雪夫距离 */
  function ChebyshevDistance(d[*,*],o[*,*]);
    outarg o;
    n=dim(d,1); m=dim(d,2);
    if m=0 then return(0);
    o_rows=dim(o,1); o_cols=dim(o,2);
    if o_rows^=o_cols or o_rows^=n then return(0);

    do r1=1 to n;
      o[r1, r1]=0;
```

```
    do r2 =r1+1 to n;
      maxdiff=0;
      do c=1 to m;
        diff=abs(d[r1,c] - d[r2,c]);
        if diff>maxdiff then maxdiff=diff;
      end;
      o[r1, r2]= maxdiff;
      o[r2, r1]=o[r1, r2];
    end;
  end;
  return (1);
endsub;

/* 距离4：闵氏距离：p=1 曼哈顿，p=2 欧氏，2->无穷切比雪夫距离 */
function MinkowskiDistance (d[*,*],o[*,*],p);
  outarg o;
  n=dim(d,1);m=dim(d,2);
  if m=0 then return(0);
  o_rows=dim(o,1); o_cols=dim(o,2);
  if o_rows^=o_cols or o_rows^=n then return(0);

  do r1=1 to n;
    o[r1, r1]=0;
    do r2=r1+1to n;
      sum=0;
      do c=1 to m;
        diff=d[r1,c] - d[r2,c];
        sum=sum+ abs(diff) ** p;
      end;
      o[r1, r2]= sum ** (1/p);
      o[r2, r1]=o[r1, r2];
    end;
  end;
  return (1);
endsub;

/*相似性：余弦相似性系数 */
function CosineSimilarity (d[*,*],o[*,*] );
  outarg o;
  n=dim(d,1); m=dim(d,2);
  if m=0 then return(0);
  o_rows=dim(o,1); o_cols=dim(o,2);
  if o_rows^=o_cols or o_rows^=n then return(0);

  do r1=1 to n;
    do r2=r1+1 to n;
      sum=0;sum2=0;sum3=0;
      do c=1 to m;
        sum=sum  + d[r1,c] * d[r2,c] ;
        sum2=sum2+ d[r1,c] * d[r1,c] ;
        sum3=sum3+ d[r2,c] * d[r2,c] ;
      end;
      if sum2=0 or sum3=0 then return(0);
      o[r1, r2]= sum / (sqrt(sum2) * sqrt(sum3));
      o[r2, r1]=o[r1, r2];
    end;
    o[r1, r1]=0;
  end;
  return (1);
endsub;

/*相关性：皮尔逊相关系数 */
```

```
  function PearsonR(d[*,*],o[*,*]);
    outarg o;
    n=dim(d,1); m=dim(d,2);
    if m=0 then return(0);
    o_rows=dim(o,1); o_cols=dim(o,2);
    if o_rows^=o_cols or o_rows^=n then return(0);

    array mean_col[1] /nosymbols;
    call dynamic_array(mean_col, m);
    do row=1 to n;
      sum=0;
      do c=1 to m;
        sum=sum+d[row,c];
      end;
      mean_col[row]=sum/n;
    end;
    do r1=1 to n;
      do r2= r1+1 to n;
        sum=0;sum1=0; sum2=0;
        do c=1 to m;
          sum =sum +(d[r1,c] -mean_col[r1]) * (d[r2,c]-mean_col[r2]);
          sum1=sum1+(d[r1,c] -mean_col[r1]) * (d[r1,c]-mean_col[r1]);
          sum2=sum2+(d[r2,c] -mean_col[r2]) * (d[r2,c]-mean_col[r2]);
        end;
        if sum1=0 or sum2=0 then return(0);
        o[r1, r2]=sum / sqrt( sum1 * sum2);
        o[r2, r1]=o[r1, r2];
      end;
      o[r1, r1]=1;
    end;
    return (1);
  endsub;

  /*相似性：Jaccard 相似性系数*/
  function JaccardSimilarity(d[*,*],o[*,*]);
    outarg o;
    n=dim(d,1); m=dim(d,2);
    if m=0 then return(0);
    o_rows=dim(o,1); o_cols=dim(o,2);
    if o_rows^=o_cols or o_rows^=n then return(0);

    do r1=1 to n;
      do r2=r1+1 to n;
        p=0; q=0; r=0;
        do c=1 to m;
          if d[r1,c] = d[r2,c] then do;
            if d[r1,c]=1 then p=p+1;
          end;
          else do;
            if d[r1,c] = 1 then q=q+1;
            if d[r2,c] = 1 then r=r+1;
          end;
        end;
        o[r1, r2]= p / (q+p+r);
        o[r2, r1]=o[r1, r2];
      end;
    end;
    return (1);
  endsub;
run;
quit;
```

第26章

神经网络

既有科学研究表明，人类大脑包含约1000亿个神经元细胞，神经元细胞之间相互连接组成神经网络。神经网络的基础是神经元细胞，但它们并不直接接触，神经元之间有一个称为“突触”的部分彼此隔离（见图26-1）。大脑工作时，各神经元处理信息后会通过这个连接网络进行信息交换，即电化学信号传递。如果两个神经元之间的“连接”足够强烈，电化学信号就会从一个细胞体的轴突传递到另一个神经元的树突；树突收集其他神经元的信号输入，只有当信号达到特定阈值时，神经元才会输出一个响应信号，并通过轴突将它传递给其他神经元，从而实现信号在神经元网络中的传播。

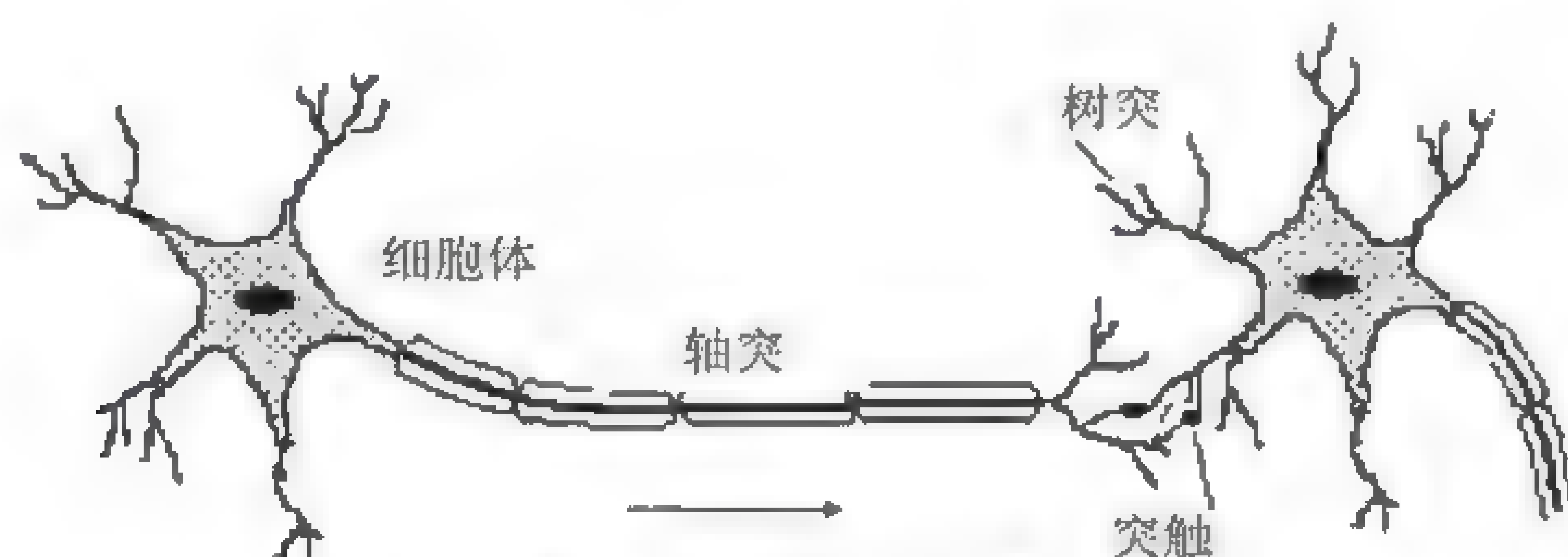


图26-1　神经元示意图

人类之所以能够思考，就是因为神经元之间构成的网络能够对外界刺激做出反应。外部刺激通过神经末梢转化为电化学信号，而电化学信号可通过轴突传递给别的神经元做进一步处理，直至输出最终控制信号对外部产生反应。

计算机领域的神经网络又称人工神经网络，它起源于用计算逻辑模仿人类大脑的工作方式。在讲述本章神经网络之前，请简单回顾一下人工神经网络的简要发展历史，它大致包括20世纪60年代的控制论、80年代中期的联结主义以及2006年以来的深度学习。

1943年，神经科学家Warren McCullock和逻辑学家Walter Pitts在合作论文中基于数学和阈值逻辑算法提出以他们名字首字母命名的第一个人工神经元模型——MCP模型。该模型将神经元处理信息抽象为输入信号的线性加权，内积运算与二值激活等几个步骤，从而开启了机器智能研究的大门。不过需要指出的一点是，此时人类的第一台计算机(1946 ENIAC)还尚未诞生。

1957年，Frank Rosenblatt发明感知器算法（Perceptron Algorithm），它使用MCP模型对输入的多维数据进行二分类，且能够使用梯度下降方法从训练样本中学习并修正神经网络权重。感知器算法是单层感知器，该算法于1962年在理论上被证明能够收敛，

引发神经网络的第一次浪潮。1959年，MIT人工智能项目联合创始人John McCarthy首次提出了“人工智能”一词，但1969年美国人工智能先驱、图灵奖获得者Marvin Minsky在其著作中证明感知器本质上是一种只能处理线性分类问题的线性模型，它甚至不能处理逻辑抑或“XOR”这一问题而导致神经网络研究近20年的低潮期。

1986年，Geoffrey Hinton发明能够用于多层感知器（Multi-Layer Perceptron, MLP）的反向传播算法（Backward Propagation Algorithm，BP算法），它采用Sigmoid函数作为激活函数进行非线性映射，突破了神经网络的线性模型局限，解决了非线性映射和学习问题。1989年，加州大学的Robert Hecht-Nielsen证明了多层感知器的万能逼近定理：对于任何闭区间内的一个连续函数都可以用包括一个隐藏层的*BP*网络来逼近。现已知3层（包括输入层）的神经网络可表示任意布尔函数；当隐藏层使用Sigmoid或其他非线性单元，输出层使用线性单元时，3层的神经网络可逼近任意一个连续函数，此时隐藏层单元的个数则取决于待逼近的函数；如果再增加一个使用Sigmoid单元的隐藏层，即4层的神经网络可对任意函数在局部以任意精度逼近。1989年Yann LeCun发明了卷积神经网络LeNet并在数字识别项目实践中取得较好效果，但后来由于1991年BP算法被指出在误差反向传递过程中存在梯度弥散问题——即乘性方式叠加到前层时，Sigmoid函数因饱和性会导致本来就很小的梯度传递到前层时几乎为零，因而无法对前层进行有效的学习。梯度弥散问题导致神经网络领域的研究再次降温。

2006年，Hiton提出使用无监督预训练对权重值进行初始化，然后采用有监督的训练微调神经网络权重的思想，成功解决了深度神经网络中的梯度弥散问题，引发了深度学习的快速发展时代。2011年能有效抑止梯度消失问题的ReLU激活函数被提出，微软首次将深度学习应用于语音识别取得重大突破，引发了深度学习（Deep Learning）的研究热潮。2012年，Hinton课题组通过采用完全有监督的学习方法构建卷积神经（CNN）网络AlexNet，参加ImageNet图像识别大赛充分证明了深度学习的极大潜力，使卷积神经网络大放异彩。2015年，Hinton、LeCun，Bengio等论证（但尚未严格理论证明）了局部极值对深度学习网络的影响在批量梯度下降优化方法中作用非常有限，结合GPU运算和并行计算在工程实践中使得深度学习研究持续火热。但时至今日，如何从少量数据中生成良好的人工神经网络依然是一大挑战。

研究神经网络有两个重要原因：一是了解人类大脑的工作机理；二是尝试构建能够解决一些传统操作型计算机无法解决的复杂问题，尤其是不确定性问题。现在已经知道神经网络能够从大量的训练样本中学习并处理各种误差，但在神经网络发明之前，传统的计算机都工作在确定性的计算操作步骤之上，整个系统可能因单个逻辑错误导致系统无法工作。

26.1 神经元模型

最早的人工神经元模型被称为感知器（Perceptron）模型（见图26-2），它能够接

受多个输入信号 $x_1, x_2, \cdots, x_i$ 产生一个输出信号 y，感知器模拟的是生物神经元能够接受多个输入信号并在特定条件下产生一个输出信号的基本原理。

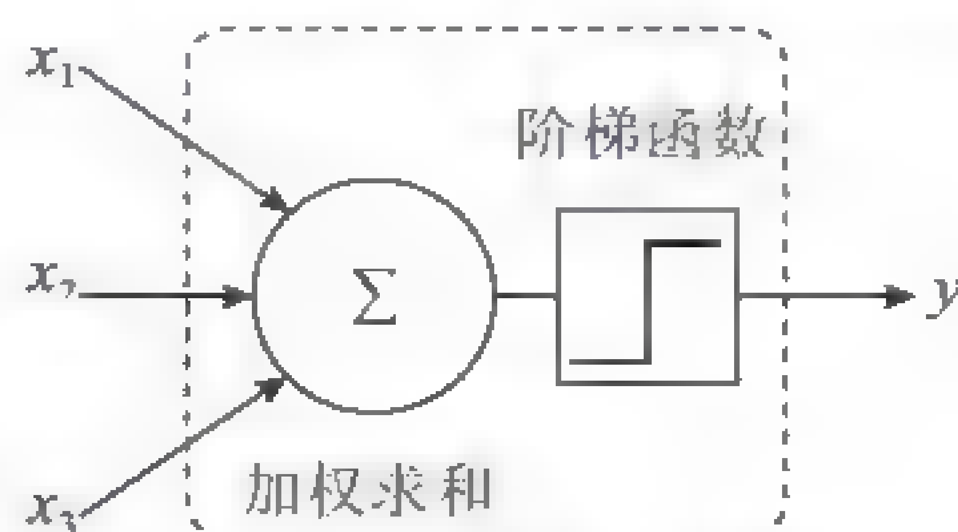


图 26-2　人工神经元（感知器）

最简单的模型就是每个输入都是 0 或者 1，分别表示某个条件成立与否，如果是 0 表示某个条件不成立，如果是 1 表示某个条件成立。比如，我们需要做一个明天去拜访朋友的决定，可以考虑如下几个因素：x_1= 朋友是否在家？ x_2= 自己是否有空？ x_3= 天气是否适合出行？而输出则只有一个：y= 去还是不去？

但更常见的情况是，各种输入的重要性是不同的。比如有些输入条件是决定性因素，而另外有些条件则是可选因素。因此在思考综合各种输入条件做出决策时，不同的输入对结果的影响不一样的。如果权重为 0 则表示对结果没有影响，如果权重是 1 表示该输入对结果影响最大。比如我们要科学选拔外贸人才，需综合考查候选人数学和英语的能力。鉴于英语是最重要的涉外沟通工具，我们设定英语的权重为 0.6，数学为 0.4 。现在某候选人 A 成绩为：英语 80，数学 90，我们可根据成绩计算出某个综合指标 z=0.6×80+0.4×90=84。而候选人 B 的成绩为英语 90，数学 80，其综合指标 z=0.6×90+0.4×80=86。现在我们根据综合指标 z 设定录取线为 85，则我们会录取候选人 B，而不是 A。录取线 85 相当于一个阈值，只要输入的综合指标大于等于阈值就输出 1 表示录取，否则输出 0 表示不录取。这个过程抽象成决策模型如图 26-3 所示。

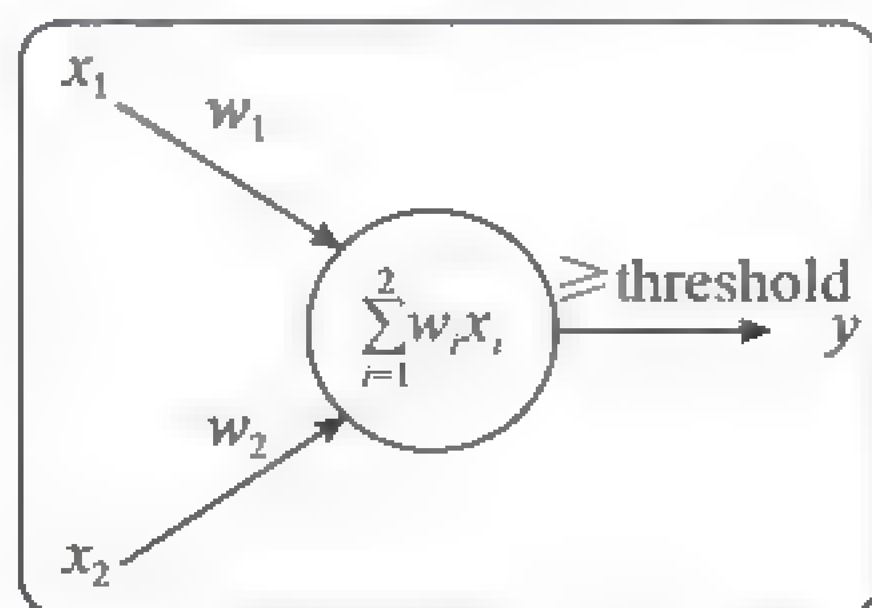


图 26-3　决策模型

实际上，上面这个感知器模型 $w_1x_1 + w_2x_2 \geq$ threshold 和 $w_1x_1 + w_2x_2 <$ threshold 反映的是由 x_1 和 x_2 构成的二维平面上的一根直线 $w_1x_1 + w_2x_2 =$ threshold 划出的两个区域。因此对于给定变量空间中的一个点 (x_1, x_2)，这个感知器可以告诉我们该点在平面上直线的哪一个区域，这些区域被称为线性可分区域。写成通用的数学表达为

$$y = \begin{cases} 0 & \text{if} \sum_i w_i x_i \leqslant \text{threshold} \\ 1 & \text{if} \sum_i w_i x_i \geqslant \text{threshold} \end{cases}$$

为进一步简化模型，我们可将输入信号 $x_1, x_2, \cdots, x_i$ 表示为向量 X，称为该感知器的输入向量，权重 $w_1, w_2, \cdots, w_i$ 表示为 W，称为感知器的权重向量；将按权重求和运算 $\sum_i w_i x_i$ 表示为 $W \cdot X$，同时将阈值 threshold 作为一个常数项考虑，假设 $b=-$threshold，则上面的数学表达可以进一步表示为

$$y=\begin{cases}0 & \text{if } W \cdot X+b<0 \\ 1 & \text{if } W \cdot X+b \geqslant 0\end{cases}$$

上面的模型就是单个感知器构成的决策模型，但现实世界中的决策模型远比这个复杂，因此需要构建一个由多个感知器节点组成的网络，称为人工神经网络。实践证明，基本布尔运算中的 AND、OR、NOT 运算都是线性可分的（见图 26-4），只需要单个感知器就足以完成判定，而 XOR 运算则不是线性可分的，它需要由多个感知器构成的“神经网络”来实现。

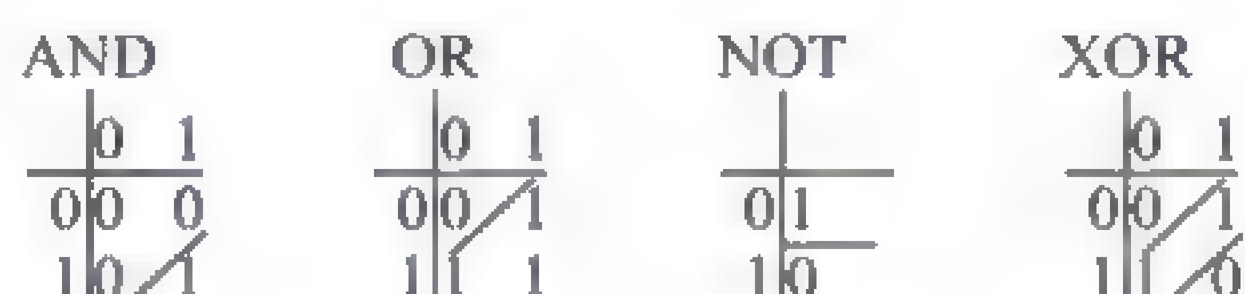


图 26-4 感知器划分线性可分区域

为统一模型，将输入信息和输出信息也作为神经网络中的一种特殊节点，用输入层和输出层表示。构成人工神经网络的神经元之间可以有多种连接形式，其中感知器分层排列的前馈神经网络（Feed-Forward Network）是最常见人工神经网络，它由一个输入层（Input Layer）、若干个隐藏层 (Hidden Layer) 和一个输出层（Output Layer）组成（见图 26-5）。其中输入层节点仅用于接收输入信号，输出层用于产生输出，而中间的隐藏层则由模仿神经元的感知器节点组成，每一层的节点可以输出信号到下一层节点，同层节点之间没有连接，而不同层节点之间的连接权重代表它们的连接强度。

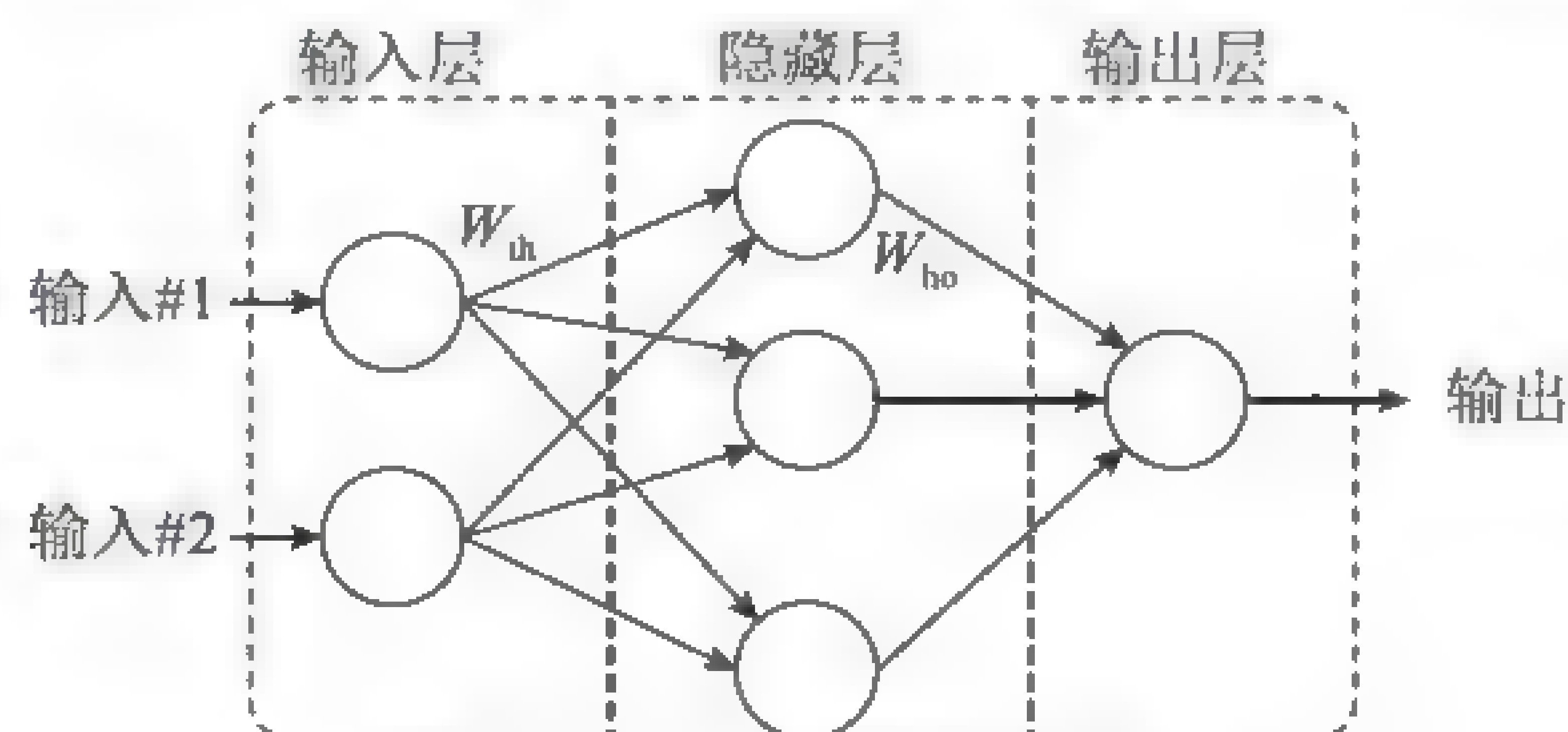


图 26-5 前馈神经网络

图 26-5 中每个感知器依然遵从多个输入信号一个输出信号的基本模式。但一个输出信号可以被后面神经网络层的多个节点所接收，成为后层节点的输入信号。现实中不是所有的信号都是逐层从前往后传播，如果后面的神经网络节点的输出会被传递给前面的节点，则构成信号的循环传递机制，则称这种人工神经网络为递归神经网络（Recurrent Neural Network），但最常见的神经网络还是前馈神经网络。

如图 26-6 所示，神经网络构建了一个输入数据 X 和输出数据 y 之间的关系，然而神经网络模型中的权重 W 以及阈值 b 依然是未知数。为了寻找恰当的权重 W 以及阈值 b，需要用已知的输入数据 X 和对应的输出数据 y 来不断调整权重和阈值，直到神经网络对于所有给定的已知数据都能满足上面的方程，这个过程通俗地称为模型训练。此时构建模型问题就变为如何调整权重 Δ_W 使得神经网络能反映从已有数据中学习到的经验值。

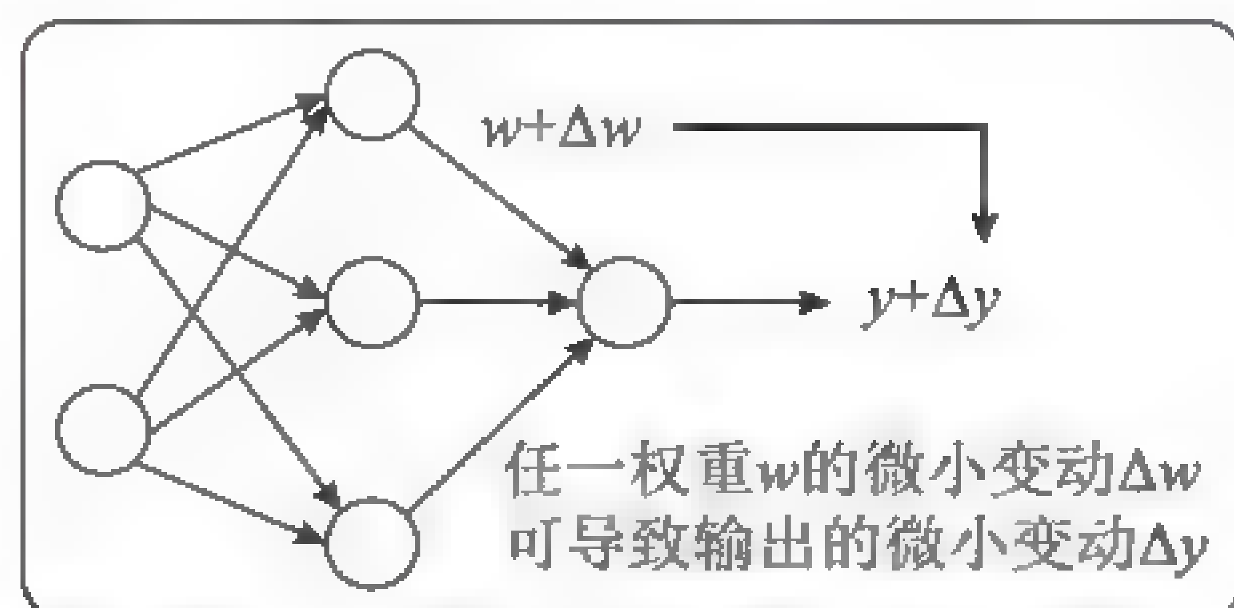


图 26-6　权重更新与输出更新的关系

从前面的探讨可知感知器的输出 y 为 0 或 1，但实际上按权重求和 $W \cdot X$ 可输出 $\{-\infty, \infty\}$ 区间上的任意实数值。为了模拟生物神经网络的阈值控制，使感知器能够输出一个非线性的输出信号，实现从连续值域到离散数值的映射，需要引入激活函数的概念。实际上，如果神经网络没有激活函数它就跟普通线性结构没什么区别，因此人工神经网络要引入非线性处理能力，激活函数是绕不开的关键一步。激活函数可以是线性函数、阈值或 S 函数，其中 Sigmoid(x) 函数最为常见。

假定感知器线性加权求和结果为 z，其中 $z=W \cdot X+b$。我们引入激活函数 $y=f(z)$

$$y=f(z)=\frac{1}{1+e^{-z}}$$

其中 z 的取值从负无穷到正无穷，而函数输出值 y 则为 0 到 1 之间的一个值，恰好与概率值的值域 [0, 1] 匹配。如果 z 取值趋向于正无穷∞，激活函数取值趋向于 1，表示输出信号非常强烈；如果 z 取值趋向于 $-\infty$，则激活函数取值趋向于 0，表示信号非常微弱。Sigmoid 函数 $y=f(z)=1/(1+e^{-z})$ 的函数曲线如图 26-7 所示。

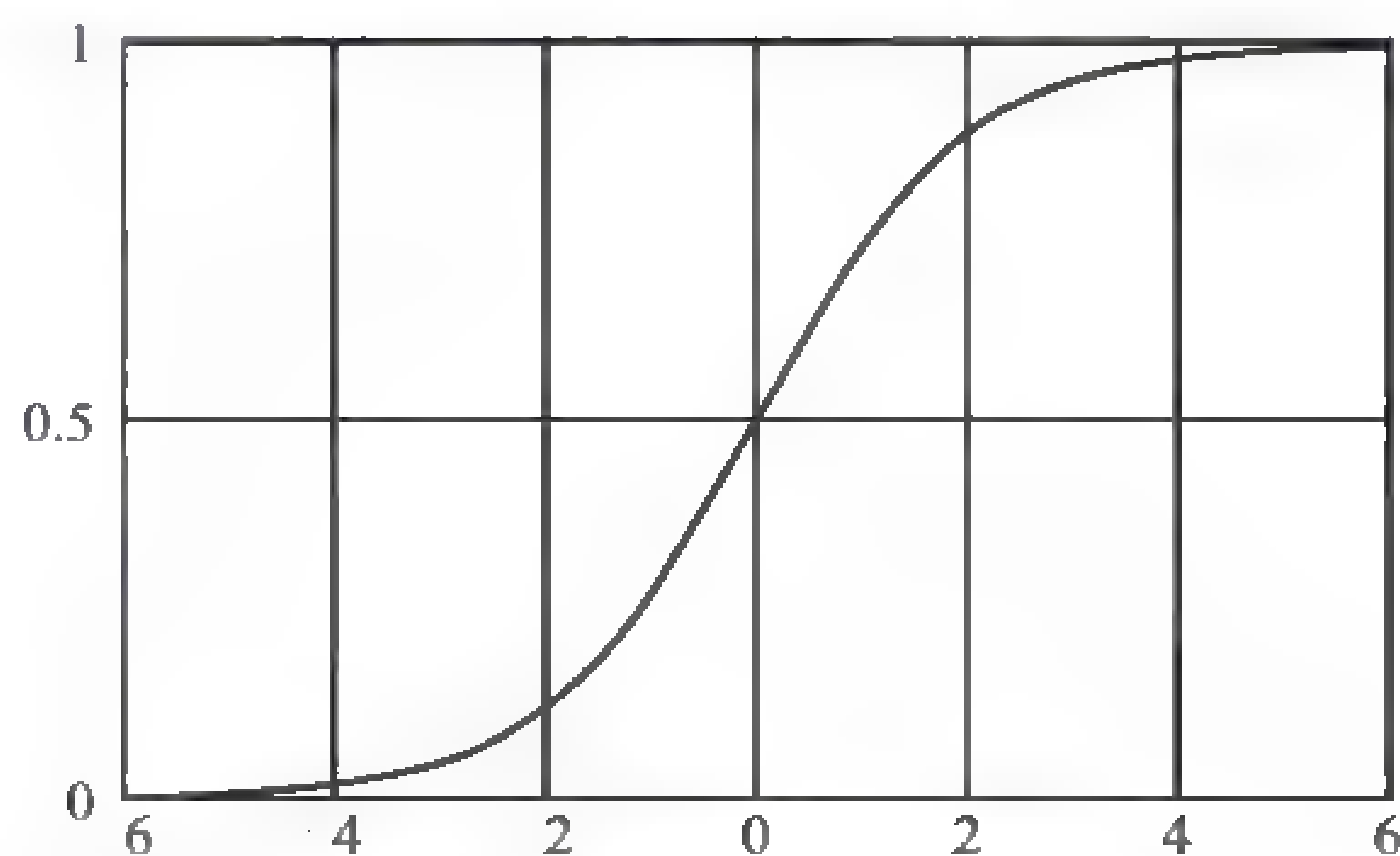


图 26-7　Sigmoid 激活函数

Sigmoid 函数有近乎线性和非线性的特性，由于该曲线函数的输出值落在 (0, 1) 之间

而被经常作为阈值函数使用于神经网络。本质上，激活函数就是将神经网络中前一层的输出累积后，将数值投影到 0~1 之间。对于 Sigmoid 函数，数学上已经证明其输出的微小改变量与权重和阈值的微小改变量之间存在如下数学关系，而这种关系可采用链式法则来计算偏导数，从而实现反向误差传播机制。

$$\Delta y \approx \sum_i \frac{\partial y}{\partial w_i} \Delta w_i + \frac{\partial y}{\partial b} \Delta b$$

实际上，可使用很多别的数学函数作为非线性激活函数，如双曲正切函数 TanH、HardLim 等，其中 Tan-Sigmoid 函数 $y=f(z)=\frac{e^z-e^{-z}}{e^z+e^{-z}}$ 也比较常用。不同的激活函数可以解决不同的问题，如果神经网络用于线性函数逼近，输出层节点使用线性函数 $y=f(z)=x$ 即可，如果神经网络用于分类，一般会使用 Sigmoid 函数。编程中需要注意激活函数在偏导数计算和反函数计算中的不同。图 26-8 反映了内积之后利用 Sigmoid 进行非线性激活的感知器模型。

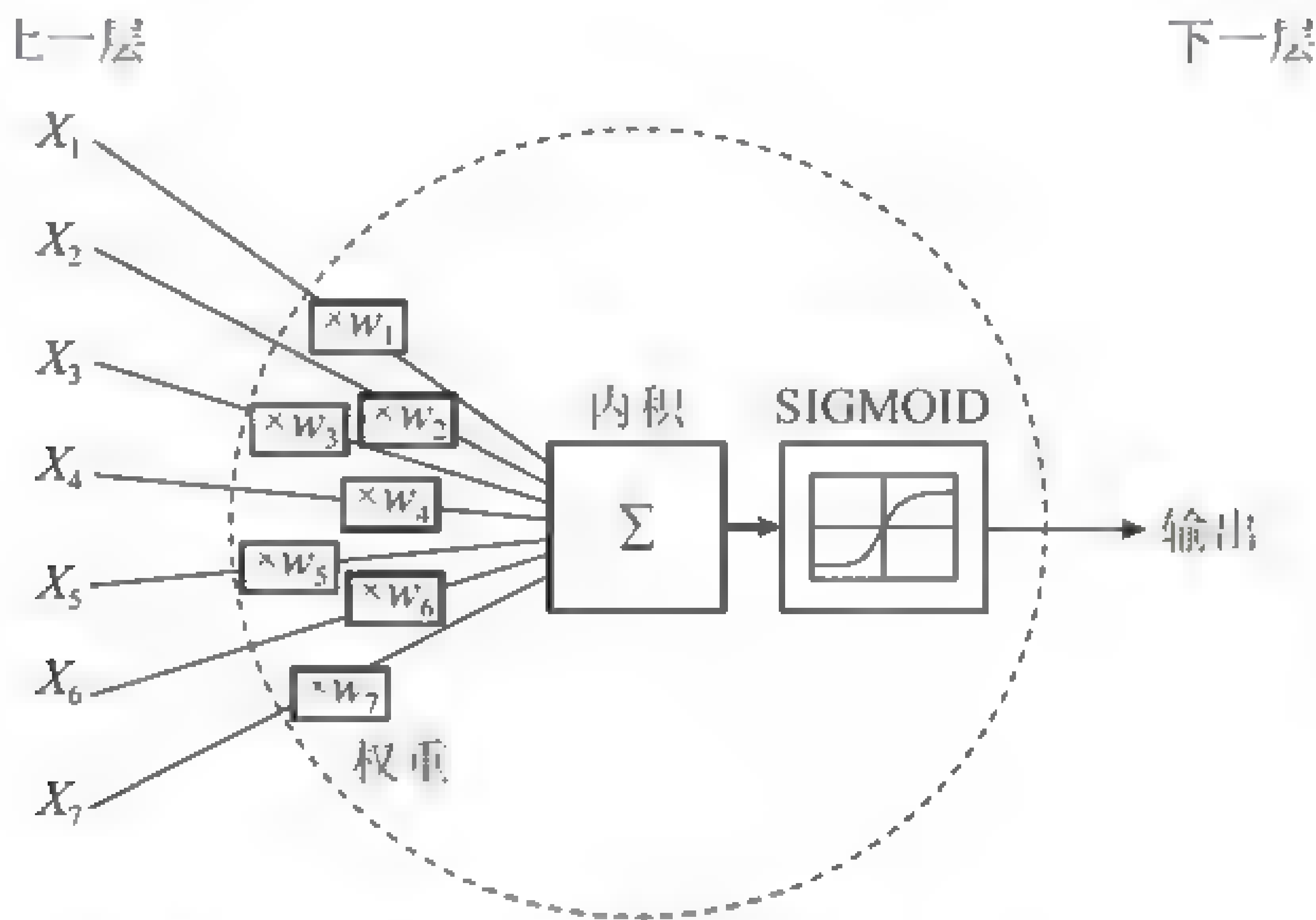


图 26-8 带 Sigmoid 激活函数的人工神经元

26.2 神经网络

对于一个神经网络，通过对已知训练数据进行学习，当训练数据量足够多时，网络就能积累足够多的“经验”。然后利用该网络就能对未知输入数据进行有效的分类或聚类，甚至发现数据中蕴含的新趋势。神经网络能够对训练数据进行两种方式的学习：有监督的学习和无监督的学习，有监督的学习就是训练数据集中的输出结果已预先知道；而无监督学习则是输入数据对应的输出结果完全未知。

对于输出结果已知的有监督学习，则往往对前馈神经网络利用反向传播算法进行处理，用于对数据进行分类；有监督的意思是期望的输出已经知道，神经网络利用它对模型参数进行不断修正。而输出结果未知的无监督学习，则对期望的输出并不知道，因此无法监督网络的迭代，此时一般采用诸如 K- 均值算法等构造网络进行评估，然后不断

迭代以找到更好的局部最优解，它可用于对数据进行聚类。

反向传播算法是 Hinton 在 1986 提出的，直到今天它依然是最流行的神经网络算法之一，其理念是利用输入数据根据 BP 神经网络算出结果，将计算结果与已知实际结果进行评估，如果结果不理想（即误差不够小）则不断调整连接权重直到模型计算结果和实际结果的误差足够小为止。

比如前面图 26-5 展示的简单神经网络结构，它由 3 层组成：输入层包含 2 个节点，隐藏层包含 3 个节点，输出层则只包含 1 个节点。输入层节点和输出层节点之间并不直接连接，而是通过一个或多个隐藏层节点进行连接。每个节点之间的连接表现为一个权重值 W。训练数据集中的输入值用于给输入层提供输入信息。比如平面上我们采集了若干个点集 (x, y)，每个点都是一个输入样本，而实际输出类别 z 也是已知，则我们可以把一个输入点 (x, y) 数据和对应的输出数据类别 z 作为一条训练样本，提供给神经网络进行训练。

神经网络实践中的主要工作包括神经网络层数的选择，各层神经元数目的选择，以及如何设置神经元之间的权重，训练神经网络参数并评估输出结果和实际结果的误差。其中还涉及前面提到的激活函数，以及学习率和动量项等概念。

（1）神经网络层数和节点数的选择取决于具体问题，比如对于一个具有 2 列输入信息和 1 列输出结果的样本，其输入层和输出层节点数就是 2 和 1；前面的神经网络就能满足要求。也就是说神经网络的输入层和输出层节点数是取决于待求解的问题和训练数据。

（2）隐藏层的层数以及每层的节点数往往取决于我们希望网络需要达到的学习深度。有时我们经常说的“深度学习”，其深度就是隐藏层数较多，每层的节点数也比较大。如果神经网络每层的节点数庞大，会导致计算量增加；而过少则有可能导致算法的学习能力不足。因此神经网络需要不断调节来保持平衡和合理的网络结构，以确保神经网络的输出具有足够改善。本质上，隐藏层节点个数就反映特征空间的维数（一个神经元能够做出单个线性划分），当具有多个隐藏层时，我们可将它理解为特征的特征。一个简单的经验公式可以确定隐含层节点的数目：$N_h = \sqrt{N_i + N_o + \alpha}$，其中 N_h、N_i、N_o 分别表示隐藏层，输入层和输出层的节点数，而 α 为调节因子，通常取值为 1~10。

（3）神经网络节点之间的权重设置与调整是神经网络训练的关键。一开始每个权重可以是随机数，也可以是根据经验人为指定的合理数值。然后通过提供训练样本的误差反馈进行不断调整。权重调整应该是连续地从整体上调整，而不是每次调整一个，每次迭代后，如果神经网络比上一次迭代能够产生更好的结果，则权重被保留并继续迭代。找到使输出数据和实际数据误差最小化的权重组合是权重更新的主要目的，也是模型训练不断趋向于“更优化”的重要指导原则。

（4）学习率（Learning Rate）：学习率是我们每次修正权重的改变量 Δw_{ji}^{l} 除以节点值 x_{lj}，再除以输出值与期望值的误差 $\mathrm{Err}(x_{l+1,i})$。早期的神经网络并没有学习率和动量项的概念，这是神经网络发展中为抑制零梯度问题引入的改进。训练神经网络时学习率数值的指定是很困难的事情，太小则需要更长时间才能收敛，太大则可能有相反效果并且导致发散。一般情况下，我们在训练神经网络时将学习率设为最常用的经验值 0.35，但某些神经网络也可以对每个连接权重有特定的学习率。

（5）动量项（Momentum Term）：神经网络中动量项的引入是为了加速算法收敛，以让迭代时权重的更新部分依赖于上一次迭代的权重值，它在某种程度上加大了搜索步长，从而能更快收敛。它的取值范围为 [0, 1]，但如果取值太大则会增大上一时刻的权重调整对当前时刻误差的梯度下降调整的影响。某种意义上动量项代表着惯性，实践中一般设置为 0.1~0.8 左右。

26.2.1 训练神经网络

训练神经网络包括两部分计算过程：输入数值的正向传递和结果误差的反向传递。

（1）根据现有的神经网络权重，逐层计算输出节点值，最后将输出层输出值跟期望值进行比较，算出误差。

（2）结果误差的反向传递时，误差用来修正神经网络中的权重参数，目的是减少神经网络输出的误差。

正向和反向传递不断重复，直到输出值和期望值的误差低于某个指定水平为止。这个训练过程本质上就是一个不断尝试找到某个局部最优的神经网络参数，使基于训练集中的所有输入用模型计算出来的输出和实际训练数据集中的期望输出之间的总体误差最小的过程。这就是为什么有人说神经网络就是一个试错网络。

从这个过程中可以看出，神经网络的能力依赖于神经网络的结构设计，初始权重参数，误差反向传播时的权重修正方法，以及训练数据样本的大小。神经网络的设计是一个不断调优和修正的过程，而训练数据的大小则在一定程度上决定了既定神经网络最终的分类能力。实践中为了科学地训练神经网络模型，一般情况下，我们可以将某个观测数据集的样本进行重新采样，如其中 K 个样本用于训练，而剩下的 L 个用于测试模型的效用，其中 $K+L=N$ 构成总的样本数（见图 26-9）。

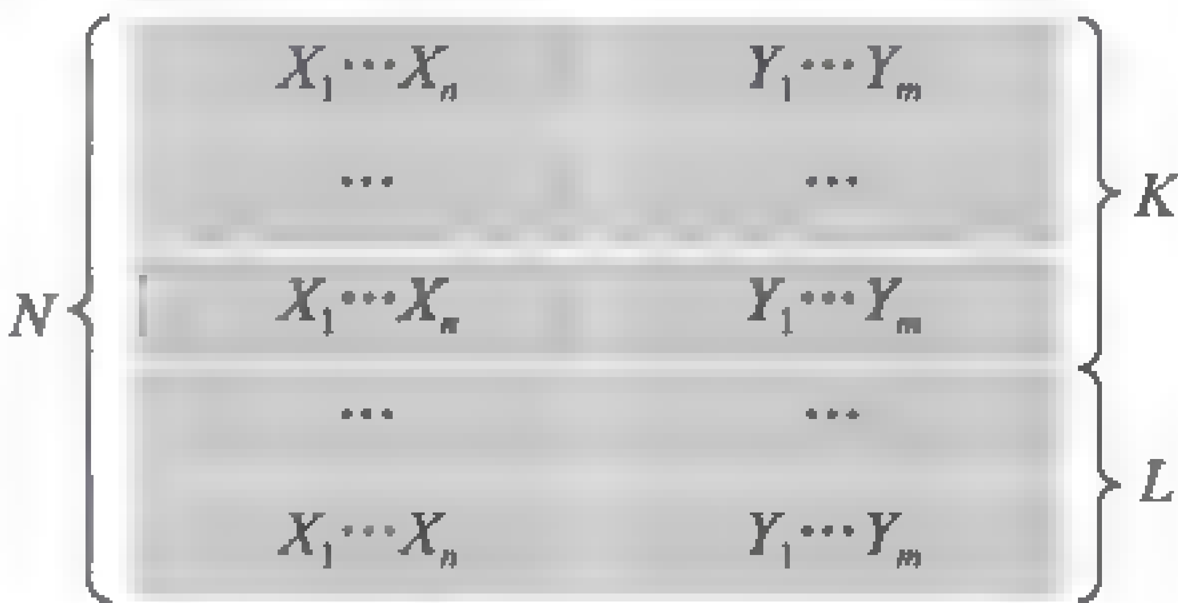


图 26-9 样本划分用于训练网络

另外，当神经网络比较庞大时，我们还需要裁剪一些无用的节点来减少神经网络的计算负载。简单的神经网络固然易于对数据进行解释，但裁剪神经网络也需要特别小心，以防止将模型过于简化而影响神经网络的效用。

26.2.2 反向传播算法

反向传播算法包括 4 个主要步骤，其核心思想是随机选择初始神经网络权重后，利

用反向传播算法不断修正神经网络的权重参数，直到神经网络的输出值和实际值的误差变得足够小。正向传播的是输入数据或信号，而反向传播的是误差。

（1）前馈计算：神经网络从训练数据中获得输入层数据，然后逐层计算隐藏层节点的值，基于这些值逐层向后计算，直到获得输出层的值。

（2）反向传播到输出层：将步骤（1）获得的输出层的值跟期望值进行比较，计算出误差，然后将误差反向传播到输出层。

（3）反向传播到隐藏层：将步骤（2）的误差在隐藏层内由后向前逐层反向传播到第一个隐藏层。

（4）不断更新权重，这一步贯穿整个算法。

下面我们以手动计算实例来说明神经网络的具体工作过程，然后用 SAS 语言实现一个完整的反向传播神经网络。为了说明问题并能够手动计算每一步，我们设计了一个尽可能简单的神经网络，目的是训练它能识别“逻辑与” 功能，即两个条件同时为真的时候， 结果才可能为真，也就是生活中常说的 “并且” 关系，其真值表如表 26-1 所列。

图 26-1 逻辑与的输入输出

输入 #1	输入 #2	输出
假 0	假 0	假 0
假 0	真 1	假 0
真 1	假 0	假 0
真 1	真 1	真 1

为此，我们设计如下人工神经网络结构：输入层为 2 个节点（因为有两个输入），隐藏层为 2 个节点，输出层为 1 个节点（因为只有 1 个输出）。神经网络的初始权重是人为指定的，为演示我们简单指定为 0.1~0.6（见图 26-10）。

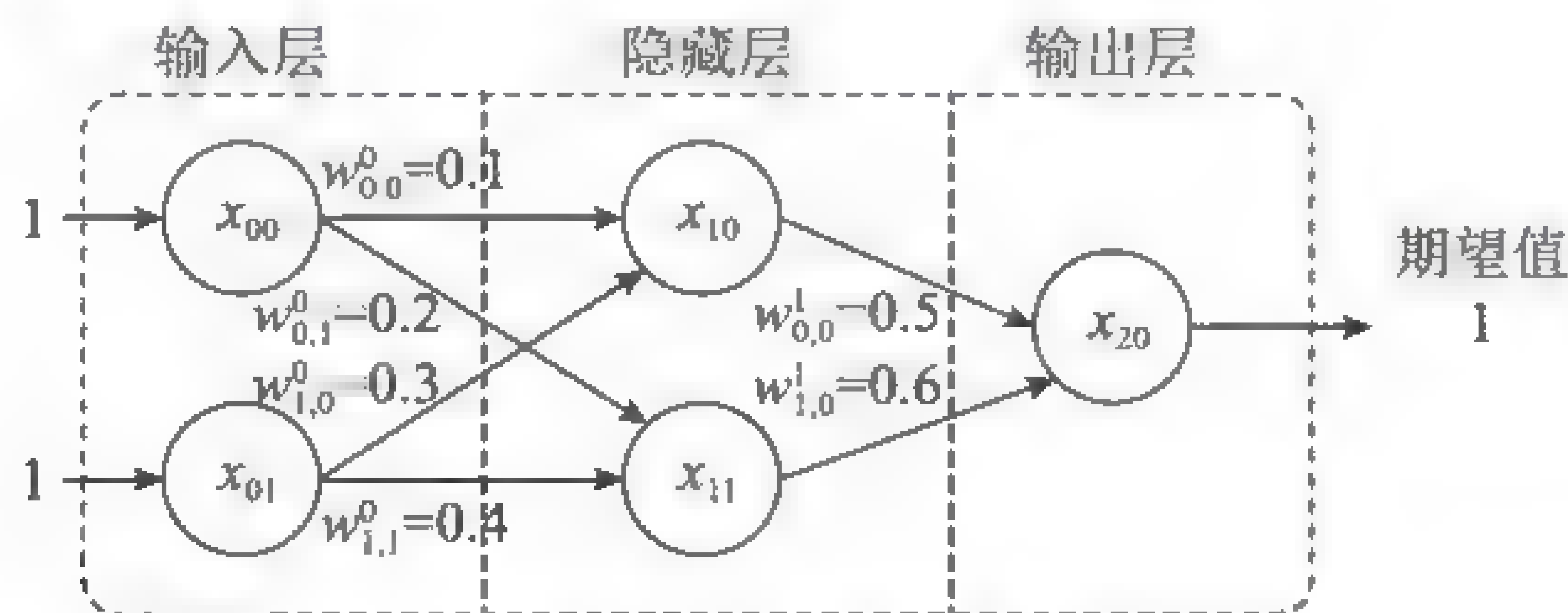


图 26-10 实现逻辑 AND 运算的神经网络

1. 前馈计算

前馈计算只涉及隐藏层和输出层，输入层数据用于接收外部数据，在神经网络训练过程中不会变化。这里我们选择简单的 S 函数作为激活函数 $f(z)=1/(1+e^{-z})$。对于每一层节点的值 x_{lj}（其中 l 表示神经网络层编号，j 表示神经网络层的节点编号，下标从 0 开始算起），其计算方法是该节点的所有输入节点乘以各自的权重求和，然后利用 Sigmoid 函数变换得到，即

$$f(z)=\frac{1}{1+e^{-z}}$$

其中每一个节点值 z 为

$$z=X_{lj}=\sum_{j=0}^{C_{l-1}}\left(x_{l-1,j}\cdot W_{ji}^{l-1}\right)$$

式中：C_{l-1} 为第 $l-1$ 层（即第 l 层的前一层）所拥有的节点数；W_{ji}^{l-1} 为第 $l-1$ 层第 j 个节点到下一层第 i 个节点之间的权重。

前面的例子中，我们为了计算方便将“假”用数字0表示，“真”用数字1表示。并且我们人为指定神经网络的初始输出权重为0.1、0.2、0.3、0.4（输入层）和0.5、0.6（隐藏层）。假定我们现在训练第1条数据（$x_{00}=1, x_{01}=1$），则隐藏层节点 x_{10} 和 x_{11} 可以如下计算：

$$x_{10}=f(x_{00}\times w_{0,0}^{0}+x_{01}\times w_{1,0}^{0})=f(1\times 0.1+1\times 0.3)=f(0.4)=0.5986876601$$

$$x_{11}=f(x_{00}\times w_{0,1}^{0}+x_{01}\times w_{1,1}^{0})=f(1\times 0.2+1\times 0.4)=f(0.6)=0.6456563062$$

一旦某个隐藏层节点被计算完毕，我们就可以正向传播：利用同样的公式和激活函数逐层计算每层的节点值，直到所有输出层节点计算完毕为止。本例中只有一个输出层节点，很容易计算出：

$$\begin{aligned}x_{20}&=f(x_{10}\times w_{0,0}^{1}+x_{11}\times w_{1,0}^{1})\\&=f(0.5986876601\times 0.5+0.6456563062\times 0.6)\\&=f(0.6867376138)\\&=0.6652408002\end{aligned}$$

2. 反向传播

计算输出层节点的误差是根据神经网络计算出来的结果和训练数据中给出的实际值进行比较得到。其计算公式为

$$\mathrm{Err}(x)=x(1-x)(x^{E}-x^{P})$$

对于 x_{20}，该节点的误差计算为

$$\mathrm{Err}(x_{20})=x_{20}(1-x_{20})(x_{20}^{E}-x_{20}^{P})$$

式中：x_{20}^{E} 为该节点期望值；x_{20}^{P} 为神经网络预测出来的值；本例中分别为1和0.6652408002。因此输出节点 x_{20} 的误差为

$$\begin{aligned}&\mathrm{Err}(x_{20})=x_{20}(1-x_{20})(x_{20}^{E}-x_{20}^{P})\\&=0.6652408002(1-0.6652408002)(1-0.6652408002)\\&=0.07454936\end{aligned}$$

输出层节点的误差一旦计算出来后，则需要用它来对隐藏层进行反向误差传播和权重调整。数学上已经证明，每个权重的梯度等于与之相连的前一层节点的输出，乘以与之相连的后一层反向传播的输出。为改善神经网络的收敛，首先需要在等式中引入动量项 α 和学习率 β。对于某个隐藏层节点的输出权重 w_{ji}^{l}，首先要找到它的改变率 Δw_{ji}^{l}，它可根据如下公式计算出来：

$$\Delta w_{ji}^{l}=\beta \cdot \mathrm{Err}(x_{l+1,i}) \cdot x_{lj}$$

对于假设的例子，其中假定设置学习率 β 为常用经验值 0.35，则隐藏层第一个节点 x_{10} 的输出权重修正量：

$$\Delta w_{00}^{1}=\beta \cdot \mathrm{Err}(x_{20}) \cdot x_{10}=0.35\times 0.07454936\times 0.5986876601=0.0156211237$$

则新的权重可根据如下公式算出，其中假定设置动量项 α 为 0.9，则新的隐藏层第一个节点 x_{10} 的输出权重为

$$w_{00}^{1}=w_{00}^{1}+\Delta w_{00}^{1}+(\alpha\times\Delta(t-1))=0.5+0.0156211237+0.9\times 0=0.5156211237$$

式中：w_{00}^{1} 为原来的输出权重；$\Delta(t-1)$ 为上一次迭代（即 $t-1$ 时刻）该节点的权重变化量。第一次迭代开始时为零，而后续迭代则为上一次迭代的权重变化量。

同理，可以算出：

$$\Delta w_{10}^{1}=\beta \cdot \mathrm{Err}(x_{20}) \cdot x_{11}=0.35\times 0.07454936\times 0.6456563062=0.0168466425$$

则新的权重 w_{10}^{1} 可根据如下公式算出：

$$W_{10}^{1}=w_{10}^{1}+\Delta w_{10}^{1}+(\alpha \cdot \Delta(t-1))=0.6+0.0168466425+0.9\times 0=0.6168466425$$

至此，我们已经将误差从输出层反向传播到隐藏层了。

对于输出层节点 x_{20}，由于预先知道了其对应的期望值 $x_{20}^{E}=1$，因此我们能够根据公式 $\mathrm{Err}(x)=x(1-x)(x^E-x^P)$ 来计算误差项 $\mathrm{Err}(x_{20})$。但对于隐藏层节点（如 x_{10} 和 x_{11}）而言，则预先并不知道它们对应的期望值，那么我们如何得到隐藏层节点的误差项呢？

由于某个节点的误差来自上一层节点的贡献，即上一层节点的误差项以其新的输出权重传播到下层节点形成了下一层节点的误差项，因此，我们需要通过权重关系逆推出上一层的误差项，即

$$\begin{aligned}\mathrm{Err}(x_{10})&=x_{10}(1-x_{10}) \cdot \mathrm{Err}(x_{20}) \cdot W_{00}^{1}\\&=0.5986876601\times(1-0.5986876601)\times 0.074554936\times 0.5\\&=0.0089556424\end{aligned}$$

$$\begin{aligned}\mathrm{Err}(x_{11})&=x_{11}(1-x_{11}) \cdot \mathrm{Err}(x_{20}) \cdot W_{10}^{1}\\&=0.6456563062\times(1-0.6456563062)\times 0.07454936\times 0.6\\&=0.0102334312\end{aligned}$$

这样，我们就得到了最后一个隐藏层的误差项，根据它以及输入的权重，就可以根据前面所述的计算方法来计算输入层节点新的输出权重：首先对每一个输出权重计算其改变量 $\Delta w_{00}^{0}, \Delta w_{01}^{0}, \Delta w_{10}^{0}$和$\Delta w_{11}^{0}$ 即 β 乘以下一层节点的误差项再乘以当前层节点的值，即

$$\Delta w_{ji}^{l}=\beta \cdot \mathrm{Err}(x_{l+1,i}) \cdot x_{lj}$$

从而得到输入层的输出权重修正量：

$$\Delta w_{00}^{0}=\beta\times \mathrm{Err}(x_{10}) \cdot x_{00}=0.35\times 0.0089556424\times 1=0.003134474$$

$$\Delta w_{01}^{0}=\beta\times \mathrm{Err}(x_{11}) \cdot x_{01}=0.35\times 0.0102334312\times 1=0.0035817009$$

$$\Delta w_{10}^{0}=\beta\times\text{Err}(x_{10})\cdot x_{01}=0.35\times0.0089556424\times1=0.003134474$$

$$\Delta w_{11}^{0}=\beta\times\text{Err}(x_{11})\cdot x_{11}=0.35\times0.0102334312\times1=0.0035817009$$

有了修正量，就可以很容易算出对应的新输出权重，它等于原来的权重加上修正量，再加上动量项 α 乘以上次迭代该权重的变化量。

$$w_{ji}^{l}=w_{ji}^{l}+\Delta w_{ji}^{l}+(\alpha\cdot\Delta(t-1))$$

则最后可以计算出新输出权重为

$$w_{00}^{0}=w_{00}^{0}+\Delta w_{00}^{0}+[\alpha\cdot\Delta(t-1)]=0.1+0.003134474+0.9\times0=0.1031344748$$

$$w_{01}^{0}=w_{01}^{0}+\Delta w_{01}^{0}+[\alpha\cdot\Delta(t-1)]=0.2+0.0035817009+0.9\times0=0.2035817009$$

$$w_{10}^{0}=w_{10}^{0}+\Delta w_{10}^{0}+[\alpha\cdot\Delta(t-1)]=0.3+0.003134474+0.9\times0=0.3031344748$$

$$w_{11}^{0}=w_{11}^{0}+\Delta w_{11}^{0}+[\alpha\cdot\Delta(t-1)]=0.4+0.0035817009+0.9\times0=0.4035817009$$

这里需要特别强调的是，计算出来的新权重不能单独更新，而是需要等所有的误差项都计算完毕后才能统一更新，否则可能用部分更新数据进行运算导致结果的无效，这需要我们在代码实现中特别注意。

至此，第一次反向传播迭代计算完毕，我们可以检验新的神经网络权重是否“更优”——即是否能更好地预测结果，预测结果的误差更小。我们只需要用新的权重计算各层节点直到输出层预测出结果为止，计算隐藏层节点值：

$$\begin{aligned}x_{10}&=f(x_{00}\times w_{0,0}^{0}+x_{01}\times w_{1,0}^{0})\\&=f(1\times0.1031344748+1\times0.3031344748)\\&=f(0.4062689497)\\&=0.6001929065\end{aligned}$$

$$\begin{aligned}x_{11}&=f(x_{00}\times w_{0,1}^{0}+x_{01}\times w_{1,1}^{0})\\&=f(1\times0.2035817009+1\times0.4035817009)\\&=f(0.6071634019)\\&=0.6472934645\end{aligned}$$

计算输出层节点：

$$\begin{aligned}x_{20}&=f(x_{10}\times w_{0,0}^{1}+x_{11}\times w_{1,1}^{1})\\&=f(0.6001929065\times0.5156211237+0.6472934645\times0.6168466425)\\&=f(0.7087529411)\\&=0.6701255469\end{aligned}$$

调整权重后神经网络计算出的值 0.6701255469 和调整前的值 0.6652408002 有一个正的改进量 0.0048847467，这离预先知道的期望值 1 更近了一点。计算其误差项：

$$\text{Err}(x_{ij})=x_{ij}(1-x_{ij})(x_{ij}^{E}-x_{ij}^{P})$$

得

$$\begin{aligned}\text{Err}(x_{20})&=x_{20}(1-x_{20})(x_{20}^{E}-x_{20}^{P})\\&=0.6701255469(1-0.6701255469)(1-0.6701255469)\\&=0.0729211554\end{aligned}$$

这说明神经网络权重更新后预测值的误差从原来的 0.07454936 减少为 0.0729211554，误差减少了 0.0016282046，说明神经网络通过反向传播算法修正后误差变小了 2.18%，神经网络利用误差反向传播修正权重后其预测的准确性确实有改进。尽管这个数值看起来很小，但不要小看这微小的改进，因为这仅仅是一次训练迭代的效果。当提高训练次数时，神经网络将会展现其神奇的自我优化能力。下面的结果为笔者用 SAS 语言实现的 3 层的 BP 神经网络，节点数分别为 2、8、1，训练 1000 次后再次对原训练数据进行分类预测，效果已极其显著（其中第 1、2 列为输入数据，第 3 列为预测值，第 4 列为期望值，即希望神经网络具有“逻辑与”运算能力的期望值）。

```
NOTE: 神经网络层数 3 { 2 8 1 } rate= 0.35 mobp= 0.9
NOTE: 训练 1000 次后神经网络预测结果如下：
x1 x2    预测值   期望值
1  1     0.97     1.00
1  0     0.02     0.00
0  1     0.01     0.00
0  0     0.00     0.00
1  2     0.99
3  4     1.00
NOTE: "DATA 语句" 所用时间(总处理时间):
实际时间            0.25 秒
CPU 时间            0.20 秒
```

26.3 SAS 代码实现与范例

为说明其工作原理，下面我们使用 SAS 的 FCMP 函数来实现可重用的反向传播神经网络算法，首先我们实现前馈计算部分，即信号或数据向前传播的过程。图 26-11 展示了数据进入神经网络输入层，直到最后计算出输出数据的完整视图。

在写程序之前，我们先约定神经网络的数据结构。首先我们定义一个二维数组 $d[n, m]$ 作为训练数据，它有 n 行 m 列，每一行数据都是一个样本。同时我们定义二维数组 $o[n, k]$，表示训练数据样本对应的输出结果。而一维数组 $c[l]$ 表示第 l 层神经网络的节点数目，x_{lj} 表示第 l 层第 j 个节点的数值，而 $w[l, j, i]$ 表示第 l 层第 j 个节点到下一层第 i 个节点的输出权重。需要特别注意，由于 SAS 语言数组下标默认从 1 开始而不是 0，这一点跟 Java 和 C/C++ 编程语言不同。

在训练神经网络之前，首先要给神经网络的权重赋初始值。实践中我们可以采用服从均匀分布在 [0, 1] 之间的随机数对它进行初始化，也可以由用户人为指定其初值（见程序 26-1）。

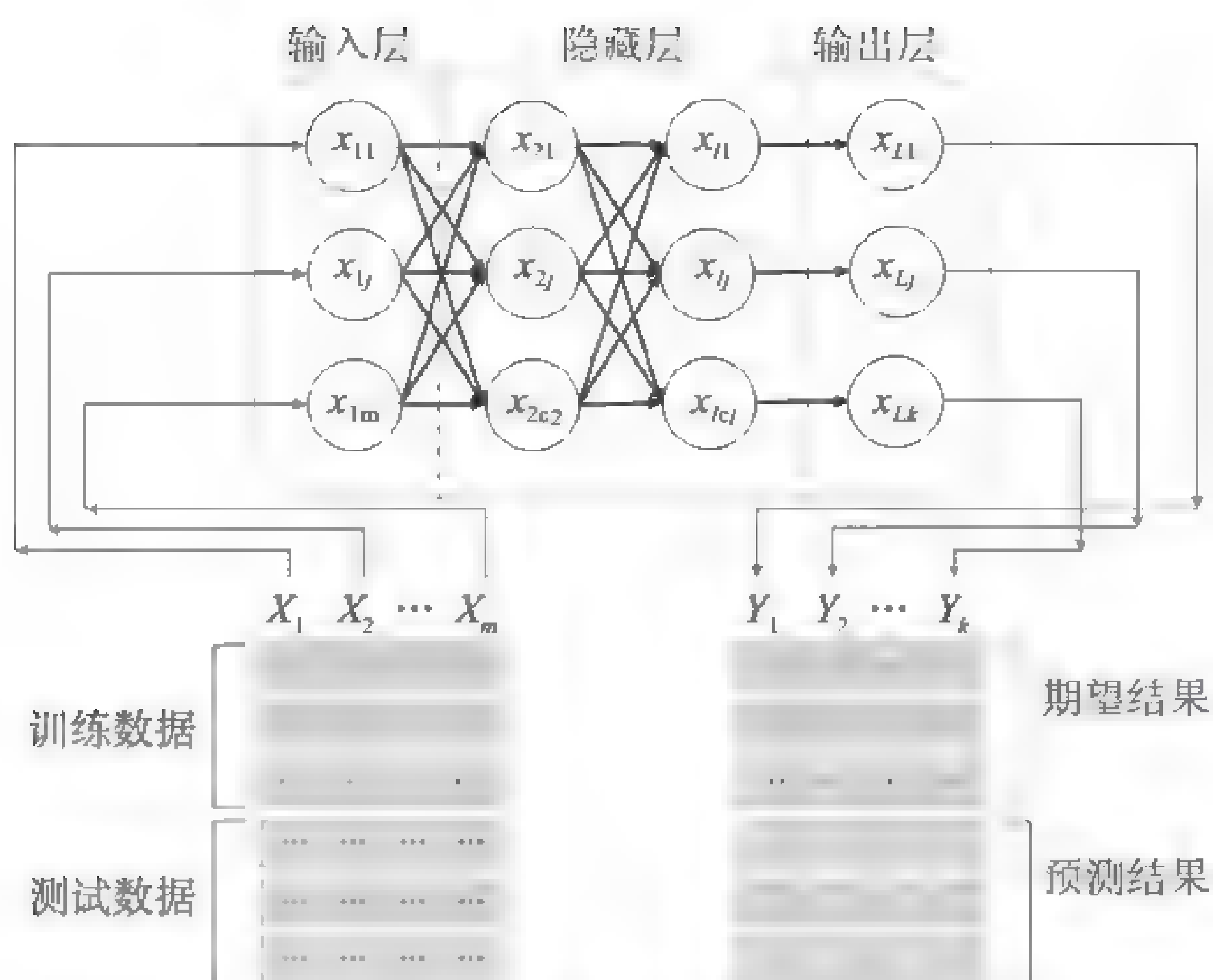


图 26-11 神经网络数据视图

程序26-1 初始化神经网络权重，包括输入层和隐藏层

```
function initweight(c[*],w[*,*,*]);
  outargs w;
  do l=1 to dim(c);
    if l <= dim(c)-1 then do;
      do j=1 to c[l]+1;
        do i=1 to c[l+1];
          w[l,j,i]=rand("UNIFORM",0,1);
          /*初始化策略：用服从 [0, 1] 均匀分布的随机数*/
        end;
      end;
    end;
  end;
  return(1);
endsub;
```

神经网络权重初始化后，就可以编写前馈计算函数 feedforward（见程序 26-2），它从第一个隐含层开始计算，逐层计算前一层节点的内积并使用激活函数得到当前层节点的值。其中要注意输入层的数据直接来自于训练数据 d，而不像隐藏层节点那样需要内积和激活函数运算，代码逻辑参见注释行。

程序26-2 前馈计算每个节点值

```
function feedforward( d [*,*], v, c[*], x [*,*], w[*,*,*]);
  outargs x;
  do l=2 to dim(x); /*从第一个隐藏层开始往前计算*/
    do j=1 to c[l];
      s=0;
      do i=1 to c[l-1];
        if l=2 then do;
          /*如果是第一个隐藏层，初值来自于输入层数据*/
          x[l-1, i]=d[v, i];
        end;
        s=s + w[l-1, i, j] * x[l-1, i];
      end;
```

```
      /*计算内积完毕，使用 Sigmoid 函数标量化 */
      x[l, j]= 1 / (1+ exp( -s));
    end;
  end;
  return( dim(x) );
endsub;
```

对于误差项的反向传播，从输出层到隐藏层逐层计算误差并修正权重，其中需要记录下权重改变量 *wd*[*l*, *j*, *i*] 以备下次循环使用。误差项的内积后需要乘上前置节点的系数 *x*[*l*, *j*]*(1−*x*[*l*, *j*])，其完整代码如程序 26-3 所示。

程序26-3　反向传播误差并修改权重

```
function backpropagation( o [*,*], v , c[*], x[*,*] , e[*,*],w[*,*,*],
                          wd[*,*,*], rate, mobp);
  outargs x,e, wd, w;

  l=dim(x);
  do j=1 to c[l];
    e[l, j]=x[l, j] * (1- x[l, j]) * (o[v, j] - x[l, j]);
    /*输出层节点的误差可以从输出数据和输出层节点的预测值直接比较得到*/
  end;

  do while( l >2 );
    l=l-1;
    do j=1 to c[l];
      se=0.0;
      do i=1 to c[l+1];
        se=se+ e[l+1, i] * w[l, j, i];
        /*对误差项进行内积*/
      end;
      e[l, j] = x[l, j] * (1- x[l, j]) * se;
      /*计算隐藏层节点的误差项*/
    end;
  end;

  l=dim(x);
  do while( l >1 );
    l=l-1;
    do j=1 to c[l];
      do i=1 to c[l+1];
        wd[l,j,i]= rate * e[l+1, i] * x[l, j] + mobp * wd[l, j, i];
        /*计算权重改变量 weight_delta */
        w[l, j, i]=w[l, j, i] + wd[l, j, i];
        /*更新权重 weight[l, j,i]*/
      end;
    end;
  end;
  return(1);
endsub;
```

至此，就可以写出训练单个样本的处理函数 train 如下（见程序 26-4）。

程序26-4　对输入数据进行训练

```
function train( d [*,*] , o[*,*], v , c[*], x [*,*], e[*,*], w[*,*,*],
                wd[*,*,*], rate, mobp);
  outargs o, x, e, wd, w;

  rc=feedforward(d, v, c, x, w);
  rc=backpropagation(o, v, c, x ,e , w , wd, rate, mobp);
  return(rc);
endsub;
```

最后，需要把上面的几个函数集成到神经网络的主函数 BPMain 中。由于 SAS 不支持锯齿状数组，需要用二维数组替代，并用一个额外的一维数组来记录各神经网络层的节点数。该函数包括如下 3 个主要步骤。

（1）分配神经网络训练所需的节点、权重、误差和变化量的数组空间，然后调用初始化权重进行初始化。

（2）利用训练样本数据 *d* 和对应的期望数据对神经网络进行重复训练，此时神经网络的权重会自动更新并逐渐优化到指定水平。

（3）训练结束后，可以用该神经网络对未知数据 testdata 进行预测和分类，并将结果写入到用户指定的输出数组 *r* 中。其完整代码见程序 26-5 所示。

程序26-5　人工神经网络主函数，支持学习率和动量项。对 testdata 进行预测输出到 r

```
function BpMain( d[*,*], o[*,*], c[*], rate, mobp, times, testdata[*,*],
                 r[*,*] );
  outargs o, r;

  /*统一用二维数组管理，找到各层最大的神经元数目*/
  max_c=constant("SMALL");
  do i=1 to dim(c);
    if c[i]>max_c then max_c=c[i];
  end;

  /*动态分配节点数组/节点误差数组（二维），权重数组/权重该变量数组(三维）*/
  array x[1,1] / nosymbols;
  call dynamic_array(x, dim(c), max_c);
  array e[1,1] / nosymbols;
  call dynamic_array(e, dim(c),max_c);
  array w[1,1,1] / nosymbols;
  call dynamic_array(w, dim(c), max_c+1, max_c);
  array wd[1,1,1] / nosymbols;
  call dynamic_array(wd, dim(c),max_c+1, max_c);
  /*为避免缺失值，我们初始化所有数组元素为 0 */
  do l=1 to dim(c);
    do i=1 to max_c; /*c[l]*/
      x[l, i]=0;
      e[l, i]=0;
      do j= 1 to max_c+1;
        w[l,j,i]=0;
        wd[l,j,i]=0;
      end;
    end;
  end;
  rc=initweight(c ,w); /*调用初始化权重处理，可根据神经网络结构需要改变*/

  /*阶段1：用输入数据 d 和 o 训练神经网络，执行 times 次学习*/
  do t=1 to times;
    do v=1 to dim(d);
      rc=train(d, o, v , c, x, e, w, wd, rate, mobp);
    end;
  end;

  /*阶段2：训练结束后，即可用训练完的神经网络进行预测分类并将结果写入数组 r中 */
  do v=1 to dim(testdata);
    l=feedforward(testdata, v, c, x, w);
    do j=1 to c[l];
      r[v, j]=x[ l, j];
    end;
  end;
```

```
  return(1);
endsub;
```

要把上面的所有函数编译到一个函数库中，只需要将前面的代码都放到SAS过程步PROC FCMP内执行，生成函数库work.funcs.bpnet即可（见程序26-6）：

程序26-6 将全部函数编译到WORK.FUNCS.BPNET 模块中

```
proc fcmp outlib=work.funcs.bpnet;
  /*将前面的所有函数放到这儿，生成 work.Funcs.bpnet 函数库 */
run;
quit;
```

最后编写该神经网络算法的测试程序，演示数据为前面我们手动计算所使用的那些数据。要在数据步内调用我们已经写好的函数，需要指定系统选项 options cmplib=(work.funcs)。其完整代码如程序26-7所示。

程序26-7 神经网络模块测试程序

```
options cmplib=(work.funcs  work.myfunc);
proc fcmp outlib=work.myfunc.BPNet;
  /*重写初始化神经网权重，如果不重写默认使用均匀分布的 0-1 随机数初始化*/
  function initWeight(c[*],w[*,*,*]);
    outargs w;
    w[1,1,1]=0.1;
    w[1,1,2]=0.2;
    w[1,2,1]=0.3;
    w[1,2,2]=0.4;

    w[2,1,1]=0.5;
    w[2,2,1]=0.6;
    return(1);
  endsub;
  run;
quit;

data _null_;
  /* 构建 3 层神经网络，节点数为2, 8, 1*/
  array c [3] _temporary_;
  c[1]= 2; c[2]=8; c[3]= 1;

  rate=0.35; mobp=0.9;/*默认学习率和动量系数*/

  /*打印神经网络基本信息*/
  ln=dim(c);
  put "NOTE: 神经网络层数 " ln '{ ' @@ ;
  do i=1 to dim(c);
    put c[i] " " @@;
  end;
  put "} rate= " rate "mobp= " mobp;

  /*准备训练数据，是一个 4 行 2 列的数据*/
  array data[4,2] _temporary_;
  data[1,1]= 1; data[1,2]= 1;
  data[2,1]= 1; data[2,2]= 0;
  data[3,1]= 0; data[3,2]= 1;
  data[4,1]= 0; data[4,2]= 0;

  /*准备训练数据期望的结果，我们模拟逻辑 AND 操作*/
  array target[4,1] _temporary_;
```

```
    target[1,1]= 1;
    target[2,1]= 0;
    target[3,1]= 0;
    target[4,1]= 0;

    /*准备检验数据：用来测试训练后的神经网络是否有效*/
    array testdata[6,2] _temporary_;
    testdata[1,1]= 1; testdata[1,2]= 1;
    testdata[2,1]= 1; testdata[2,2]= 0;
    testdata[3,1]= 0; testdata[3,2]= 1;
    testdata[4,1]= 0; testdata[4,2]= 0;
    /*加入神经网络没有学习过的数据 (1, 2) 和 (3, 4) 进行泛化测试*/
    testdata[5,1]= 1; testdata[5,2]= 2;
    testdata[6,1]= 3; testdata[6,2]= 4;

    /*用于接收神经网络预测结果，它和 target 列宽一样，但行数要跟 testdata 一样*/
    array testdata_p[6,1] _temporary_;

    /*一切妥当，训练 1000 次*/
    times=1000;
    rc=BpMain(data, target, c, rate, mobp, times, testdata,testdata_p);
    put "NOTE: 训练 " times "次后神经网络预测结果如下:";

    /*打印神经网络预测的结果*/
    put "x1 x2 预测值期望值";
    do i=1 to dim(testdata);
      do j=1 to dim(testdata,2);
        put testdata[i,j] " " @@;
      end;
      do j=1 to dim(testdata_p,2);
        put testdata_p[i,j] 6.2" " @@;
        if i<= dim(target) then put target[i,j] 6.2" " @@;
      end;
      put " ";
    end;
run;
```

运行程序 26-7 代码将得到如下结果（见图 26-12），从结果中可以看出：所有训练过的数据都能非常好地被预测（0.97 非常接近 1），甚至在没有训练过的两条数据 (1, 2) 和 (3, 4)，该神经网络也能准确预测出 1.00 和 1.00，确实非常令人惊异。

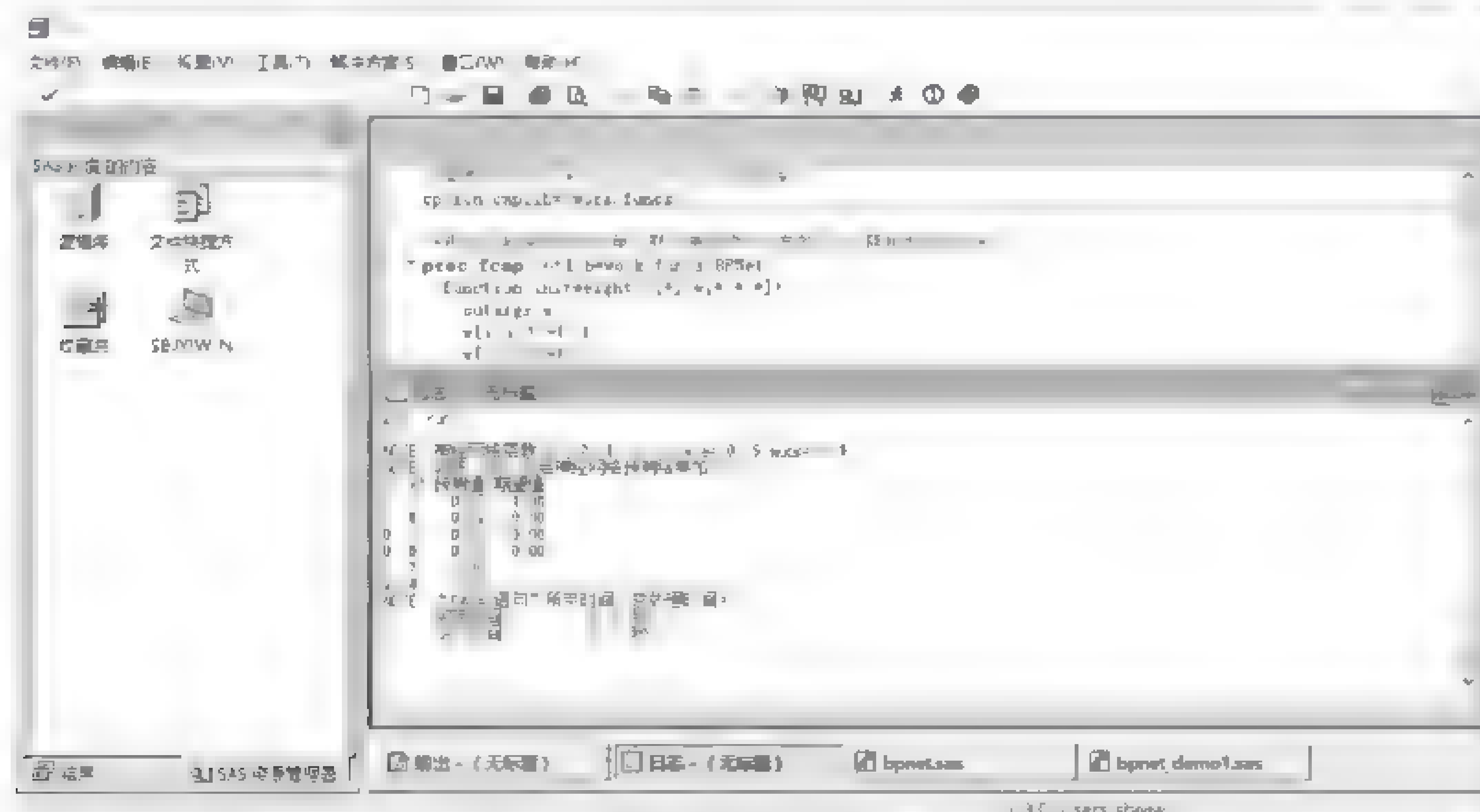


图 26-12 神经网络预测结果

实际上，读者可以利用这个通用的神经网络框架代码写出自己的BP神经网络程序。实践证明，反向传播神经网络对数据分类具有很强的非线性处理能力，如通过对如下坐标平面上的3个实心圆和2个三角形用神经网络训练10000次，再用训练出来的神经网络对坐标平面上所有的25个点进行预测，发现它确实能做出令人惊异的非线性预测结果：位于左上角和右下角被准确地预测为三角形而中间带状区域则被预测圆形（如图26-13所示，虚线为分割区域示意线）。

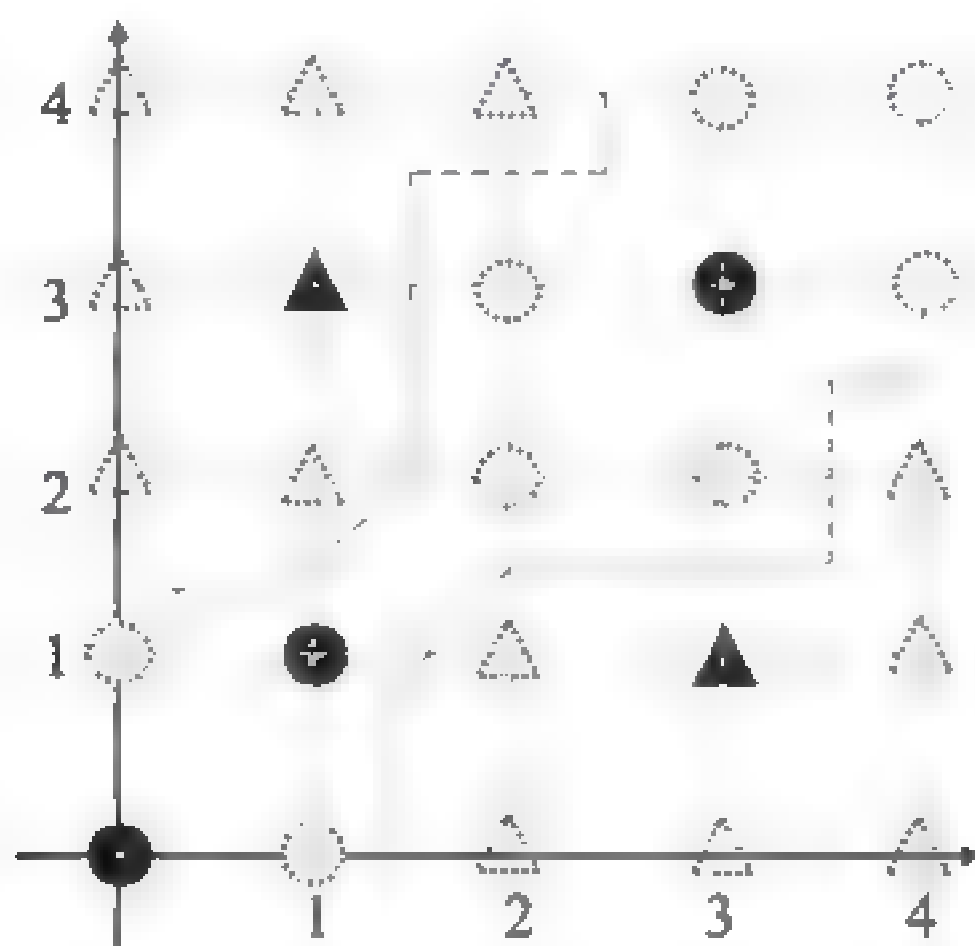


图26-13 平面形状的学习与预测

实现该平面形状的神经网络学习和预测的完整SAS代码如程序26-8所示。

程序26-8 训练5个点对平面上的 25个点进行分类

```
options cmplib=(work.funcs);
data _null_;
  /* 1) 构建 3 层神经网络 */
  array c[3] _temporary_;
  c[1]= 2; c[2]=10; c[3]= 2;

  rate=0.35; mobp=0.8;

  /*打印神经网络基本信息*/
  ln=dim(c);
  put "NOTE: 神经网络层数 " ln '{ ' @@ ;
  do i=1 to dim(c);
    put c[i] " " @@;
  end;
  put "} rate= " rate "mobp= "   mobp;

  /*2) 准备训练数据，是一个 5 行 2 列的数据*/
  array data[5,2] _temporary_;
  data[1,1]= 0; data[1,2]= 0;
  data[2,1]= 1; data[2,2]= 1;
  data[3,1]= 1; data[3,2]= 3;
  data[4,1]= 3; data[4,2]= 1;
  data[5,1]= 3; data[5,2]= 3;

  /*准备训练数据期望的结果，(1 0) 表示圆形， (0,1) 表示三角形*/
  array target[5,2] _temporary_;
  target[1,1]= 1; target[1,2]= 0;
  target[2,1]= 1; target[2,2]= 0;
  target[3,1]= 0; target[3,2]= 1;
  target[4,1]= 0; target[4,2]= 1;
  target[5,1]= 1; target[5,2]= 0;
```

```
  /*3) 准备检验数据：用来测试训练后的神经网络是否有效*/
  array testdata[25,2]  temporary ;
  do i=0 to 4;
    do j=0 to 4;
      k=(i*5+(j+1));
      testdata[k, 1]=j;
      testdata[k, 2]=i;
    end;
  end;

  /*用于接收神经网络预测的结果，它列宽和 target 一样，但行数要和 testdata 一样*/
  array testdata_p[25,2] _temporary_;

  /*4) 一切妥当，训练10000 次*/
  times=10000;
  rc=BpMain(data, target, c, rate, mobp, times, testdata,testdata_p);
  put "NOTE: 训练 " times "次后神经网络预测结果如下:";

  /*5) 打印神经网络预测的结果*/
  put "x1 x2  实际值  实际值  预测值  预测值";

  do i=1 to dim(testdata);
    /*打印验证数据*/
    do col=1 to dim(testdata,2);
      put testdata[i,col] " " @@;
    end;
    /*若是训练数据，打印实际数值*/
    match_row=0;
    do r=1 to dim(data);
      bmatch=1;
      do col=1 to dim(data,2);
        if testdata[i, col]^=data[r, col] then do;
          bmatch=0;
          leave;/*break*/
        end;
      end;
      if bmatch=1 then do;
        match_row=r;
        leave;/*break*/
      end;
    end;
    if match_row>0 then do;
      do col=1 to dim(target,2);
        put target[match_row,col] 6.2 " " @@;
      end;
    end;
    else do;
      do col=1 to dim(target,2);
        put "       " @@;
      end;
    end;
    /*打印预测数据*/
    do col=1 to dim(target,2);
      put testdata_p[i,col] 6.2 " " @@;
    end;
    put " ";
  end;
run;
```

```
        end;
      end;
      if bmatch=1 then do;
        match_row=r;
        leave;/*break*/
      end;
    end;
    if match_row>0 then do;
      do col=1 to dim(target,2);
        y[col]=target[match_row,col];
        put target[match_row,col] 6.2 " " @@;
      end;
    end;
    else do;
      do col=1 to dim(target,2);
        y[col]=.;
        put "       " @@;
      end;
    end;
    /*打印预测数据*/
    do col=1 to dim(target,2);
      z[col]=testdata_p[i,col];
      put testdata_p[i,col] 6.2 " " @@;
    end;
    put " ";
    output;
  end;
```

此时，结果数据集 Result 中包含了对结果的预测数据，其中 z1 和 z2 列表示其预测出来的形状。为直观展示预测结果，应基于结果数据集构建 Result2，其中 Shape=1 表示圆，而 0 表示三角形。随后再构建一个属性表 attr_map 用于自定义 SGPLOT 输出的散点形状。最后用 SGPLOT 过程步将平面上的 25 个点绘制出来，其完整代码见程序 26-10 所示。

程序26-10　将预测结果用SGPLOT绘图

```
data Result2;
  set Result;
  shape=(z1>=z2);/*生成形状表示 1 表示圆，0 表示三角形*/
run;;

data attr_map;/*创建属性表*/
  input id $ value $ fillcolor $ markercolor $ markersymbol $ ;
  datalines;
id1 1 white red  circle
id1 0 white blue triangle
;
run;
proc sgplot data=result2 dattrmap=attr_map;/*用属性表进行分组*/
  scatter y=x1 x=x2 / group=shape  attrid=id1;
  xaxis  grid;
  yaxis  grid;
run;
```

输出结果如下图 26-14 所示。

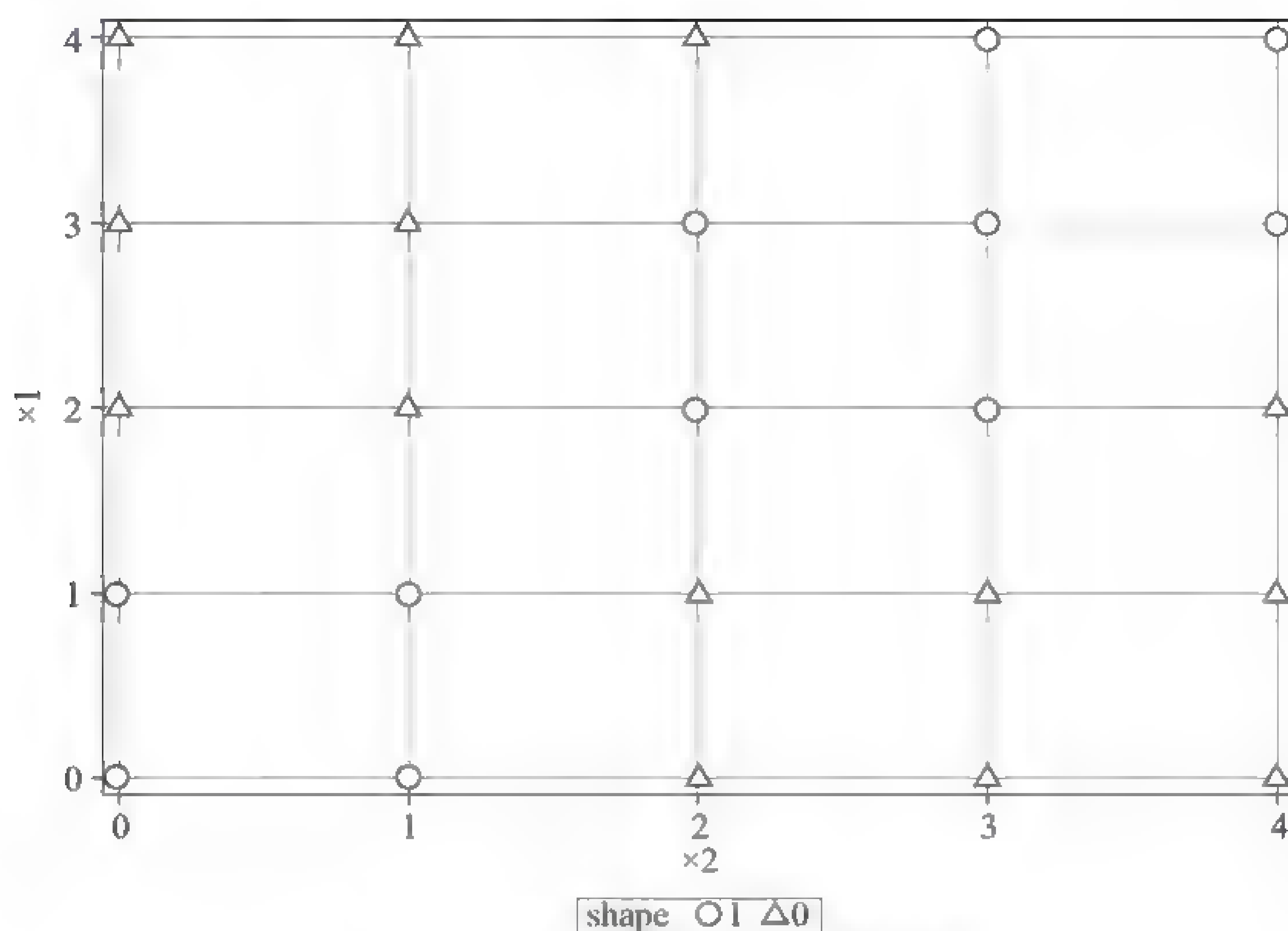


图 26-14　神经网络对形状的预测结果

实际上，神经网络领域有一套与本章之前的传统数据分析不同的术语体系，读者在阅读文献的时候要特别注意。统计学中的变量在神经网络领域被称为特征，独立变量 X 被称为输入，依赖变量 Y 被称为目标或训练值，而预测值则被称为输出值。模型参数估计被称为神经网络的训练、学习或自适应过程，数据变换也被称为函数链等。神经网络领域没有传统统计中的总体和样本的概念，而是引入了训练集、验证集和测试集等数据划分概念。像传统的判别分析和回归分析这种预先知道结果目标值 Y 的神经网络训练被称为有监督学习，而预先不知道目标值 Y 的聚类分析，其神经网络训练被称为无监督学习。

本章笔者基于程序员思路开发的神经网络模块展示了如何在 SAS 中自己开发神经网络应用，但实际上 SAS 产品线中提供许多非常专业的神经网络处理过程步，通常都需要单独的软件授权方可使用。在多层感知器模型的神经网络训练中，用到了一个支持无限极小化的非线性目标函数，而理论上至今也没有找到寻找目标函数全局最小值的确实可行办法，只能得到局部最优解，因此一般实践中需要采用不同初始值对权重进行多次训练并进行模型比较来优化选择。

SAS 系统中提供了各种专业方法对神经网络进行训练和预测，包括非线性建模和优化程序，SAS STAT 中的 NLIN 过程步，SAS/ETS 中的 MODEL 过程步，SAS OR 中的 NLP 过程步以及 SAS/IML 中的 NLP 子例程，更常见的是使用 EM 产品中的 NEURAL 过程步和最新 SAS VIYA 产品中的 NNET 过程步做神经网络应用的开发。

（1）PROC NEURAL 该过程步利用已证实可靠的统计方法和数值算法，对前馈神经网络进行训练，其中 INPUT、HIDDEN 和 TARGET 语句分别用来定义不同的神经网络层，每个神经网络层包含若干作为最小计算实体的节点或神经元。输入层和目标层节连接必须是并行的，而隐藏层则可串行或并行连接。PROC NEURAL 支持 11 种激活函数，

调用它之前需要调用 PROC DMDB 来创建一个 DMDB 编码的训练集和一个 DMDB 目录册入口，感兴趣的读者自行查看帮助文档来了解调用细节。

（2）PROC NNET 是 SAS Viya 的 SAS Visual Data Mining & Machine Learning 8.1 产品中最新提供的过程步，可用于训练多层感知神经网络，也可用于建立高性能编解码的自动编码器（AutoEncoder）以提取特征和执行非线性主成分分析等，它可很方便地构建图像识别等领域的神经网络。另外，运行在 SAS CAS 服务器上的过程步现在已经实现了开放调用接口，即编程人员可使用 Java、Python 或 Lua 语言，以及 SAS 运行环境中特别提供的 CASL 语言调用 SAS 产品中的强大分析功能。

比如下面简短的 SAS 代码（见程序 26-11）对经典的房屋贷款数据集 SAMPSIO.HMEQ 加载至 CAS 表 MYCASLIB.HMEQ 的 5960 条房屋贷款申请数据（该数据集中有个变量BAD包含一个指示贷款申请人是否在贷款批准后还清贷款还是拖欠贷款的标志）进行训练，神经网络模型训练结果被输出到 MYCASLIB.MYNNETMODEL 中用于进一步的预测。结果表明该模型的误分类率在训练集、验证集和测试集上分别为 0.20，0.18 和 0.20 左右。

程序26-11 利用NNET对房贷数据进行训练预测

```
proc nnet data=mycaslib.hmeq standardize=midrange missing=mean;
  architecture mlp;
  input job reason / level=nominal;/*输入层*/
  input debtinc delinq loan mortdue value yoj derog clage clno;
  hidden 7;/*隐藏层*/
  target bad / level=nominal; /*目标层*/

  /*优化算法为 LBFGS算法，也可指定随机梯度下降算法 SGD;最大迭代次数 500.
    LBFGS全称Limited-memory Broyden-Fletcher-Goldfarb-Shanno算法;
    SGD全称为Stochastic Gradient Descent算法*/
  optimization algorithm=lbfgs maxiter=500;

  train outmodel=mycaslib.mynnetmodel seed=12345;
  /*将数据分区为训练集70%,验证集20%和测试集 10%*/
  partition fraction(validate=0.2 test=0.1 seed=54321);
run;
```

第 17 章

π 高精度求解与探索分析

数据分析学科是一门以数据为基础，以各种分析方法为手段获取洞见（Insight）和智慧（Intelligence）的学科。它并不总是充满了令人生畏的数学公式和抽象推导过程，而是基于现实世界在数字空间的投影，构建具体认知模型的有趣学科。分析方法五花八门，相关书籍也多种多样，然而，正如著名统计学家乔治 E.P. 博克斯（George E. P. Box）在本书扉页所指出的那样：模型皆有误，或尤建奇功。其实也不难想象，基于数据构建的模型难道确实是现实世界的精确倒影吗？作为数据分析的镜子可以照见现实，其影子就是分析模型。对于数据分析学科而言，基于同样的数据，不同分析方法能得出不同的结论，至于是否与现实语义匹配，则是需要结合实际经验进行解释。

笔者曾经深思的一个问题是，这个世界上是否存在一些既恒定又不断变化的数据，它足以让我们持续地探讨其恒定背后无尽的变化呢？这必须在数学的王国中寻找它们。数学之美在于简洁而抽象的公式所隐含的无穷变化，以及某些数字所具有的奇特性质，如各种无理数，包括圆周率 π，自然对数的底数 e 以及黄金比率 φ 等。这些数字的神奇之处在于它们普遍存在于各种复杂的数学和物理公式之中，似乎是上帝对这个世界进行编程的控制点。在被称为“上帝创造的公式”——欧拉恒等式中，五个最基本的数学常数：常数 0，自然数 1，虚数单位 i，圆周率 π 和自然对数的底 e 被神秘地联系在一起：

$$e^{\pi i}+1=0$$

其中 i 的平方等于 -1，e 和 π 则是无理数常数，其无限不循环小数部分蕴含着无穷无尽的变化。π 作为人们最熟知的常数，就是任何一个圆的周长与其直径的比值，称为圆周率。公元前 225 年阿基米德（Archimedes）通过阿基米德算法求得圆周率的第一个严格近似值 223/71 (3.14048) $\leqslant \pi \leqslant$ 22/7 (3.14285)，精确到小数点后两位数，因此在西方圆周率被称为阿基米德常数。

目前人们对圆周率在数学上的确定性认识主要有：1761 年，瑞士数学家约翰•海因里希•朗伯（Johann Heinrich Lambert）证明 π 是一个无理数；1794 年，法国数学家阿德里安 - 马里•勒让德（Adrien-Marie Legendre）证明 π^2 也是一个无理数；1882 年，德国数学家费迪南德•冯•林德曼（Ferdinand von Lindemann）证明 π 是一个超越数，即 π 是不满足任何整系数多项式方程的一个实数；1953 年，库尔特•马勒（KurtMahler）证明 π 不是一个刘维尔数。

然而圆周率并不是人为创造的一个常数，而是在现实世界中自然出现的神奇数值，所有与圆或球相关的运算都会涉及圆周率，不仅如此，它还出现在标准正态分布的概率

密度函数，以及蒲丰投针试验的针与线相交的概率之中，π 的小数位包含任何一个自然数，它本身也甚至出现在所有自然数平方的倒数之和 $\sum_{k=1}^{\infty} 1/k^2 = \pi^2/6$ 之中（即巴塞尔问题）。

鉴于 π 值的独一无二的神奇数学特性，且人类至今对其分析特性了解不多，本章将探讨如何用 SAS 对无理数 π 值进行精确计算并尝试分析研究其数值规律，读者也可在此计算的基础上构建基准数据集用于分析方法和运算性能的评估。

27.1 π 值计算

圆周率的计算经过漫长的演进，经历了早期的几何方法到后来的代数方法，从割圆法手工计算演进到后来的电子计算机自动迭代运算，最终将 π 的数值计算精度推进到 2016 年 22.4 万亿位级别的历史新高度。下面简要回顾 π 值的计算历史及思想，然后用 SAS 编程实现最经典的几种计算方法。

1. 几何方法

很早以前古人就已发现，一个圆的周长和直径之比是一个与圆的大小无关的常数。比如公元前 20 世纪的埃及人就发现圆周率约为 $(16/9)^2=3.160493$，古巴比伦人估计出圆周率为 3+1/8=3.125，精确到 π 值小数点后一位数。世界公认中国古代的几何学者做得最好，主要贡献者为中国古代数学家刘徽，公元 263 年他发明割圆术为计算圆周率提供了科学方法并计算出徽率 157/50=3.14，后来又割圆至 3072 边形计算出更好的圆周率 3927/1250=3.1416。公元 480 年数学家祖冲之计算出约率 22/7 和密率 355/113，后世科学家证明这是分母小于 16604 中最接近 π 值的整数表达形式，密率的分子分母恰好由 3 个成对奇数 113355 对折而成，《隋书·律历志》明确记载祖冲之确定了圆周率的真值介于朒数 3.1415926 与盈数 3.1415927 之间，成功将圆周率推算至小数点后 7 位，从此中国在圆周率计算上的领先地位一直持续到 15 世纪。

1427 年，波斯数学家阿尔·卡西（Al-Kashi）通过分别计算圆的内接 / 外接正 32 边形的周长将 π 值计算精确到小数点后约 14 个正确值，这才打破了中国人保持上千年的纪录，后来他的领先纪录也保持了 200 年左右。

2. 韦达公式

1579 年，法国数学家弗朗索瓦·韦达（Franciscus Vieta）是最早明确地给出 π 值无穷解析式的，人类从此摆脱了依赖传统几何学对 π 进行求解的方法，进入 π 的解析求解时代。韦达给出的公式为

$$\frac{2}{\pi}=\frac{\sqrt{2}}{2}\cdot\frac{\sqrt{2+\sqrt{2}}}{2}\cdot\frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2}\cdots$$

他发现该公式的思路归功于正弦和余弦函数的倍角公式：$\sin(2x)=2\sin(x)\cos(x)$ 和 $\cos(2x)=2\cos^2(x)-1$。根据正弦二倍角公式有 $\sin(x)=2\sin(x/2)\cos(x/2)$，通过利用 $\sin(x/2)$ 代

入不断迭代即可得

$$\sin(x)=2\sin\left(\frac{x}{2}\right)\cos\left(\frac{x}{2}\right)=2^2\sin\left(\frac{x}{4}\right)\cos\left(\frac{x}{4}\right)\cos\left(\frac{x}{2}\right)=2^n\sin\left(\frac{x}{2^n}\right)\prod_{i=1}^{n}\cos\left(\frac{x}{2^i}\right)$$

将方程两边除以 x 可得 $\frac{\sin x}{x}=\frac{2^n}{x}\sin\left(\frac{x}{2^n}\right)\prod_{i=1}^{n}\cos\left(\frac{x}{2^i}\right)$。两边取极限且由于存在 $\lim_{n\to\infty}\frac{2^n}{x}\sin\left(\frac{x}{2^n}\right)=1$，所以有

$$\frac{\sin x}{x}=\lim_{n\to\infty}\frac{2^n}{x}\sin\left(\frac{x}{2^n}\right)\prod_{i=1}^{n}\cos\left(\frac{x}{2^i}\right)=\prod_{i=1}^{n}\cos\left(\frac{x}{2^i}\right)$$

此时令 $x=\pi/2$ 代入上面的方程即得

$$\frac{\sin x}{x}=\frac{1}{\pi/2}=\prod_{i=1}^{n}\cos\left(\frac{\pi}{2^{i+1}}\right)=\cos\left(\frac{\pi}{4}\right)\cos\left(\frac{\pi}{8}\right)\cos\left(\frac{\pi}{16}\right)\cdots$$

此时再根据三角函数余弦二倍角公式有

$$\cos(x/2)=\sqrt{\frac{1+\cos x}{2}}$$

$\cos(x)=2\cos^2(x/2)-1$ 推出

代入上式即得韦达公式：

$$\frac{2}{\pi}=\cos\left(\frac{\pi}{4}\right)\cos\left(\frac{\pi}{8}\right)\cos\left(\frac{\pi}{16}\right)\cdots=\frac{\sqrt{2}}{2}\frac{\sqrt{2+\sqrt{2}}}{2}\frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2}\cdots$$

尽管韦达将圆周率求解精度推进到小数点后 9 位数，其方法本质上还是多边形法。后世数学家依然在计算圆周率上耗费了大量的精力，如德国数学家鲁道夫·范·科伊伦（Ludolph Van Ceulen）几乎耗尽一生的时间采用阿基米德割圆法计算圆周率，并在 1609 年获得了圆周率 35 位精度值，即介于 3.14159 26535 89793 23846 26433 83279 50288 和 3.14159 26535 89793 23846 26433 83279 50289 之间。他对该成就感到非常自豪因此将 35 位圆周率刻在自己墓碑上，而圆周率至今在德国依然被称为鲁道夫数。然而以上这些方法主要是几何求解方法，其计算效率和精度有限。

3. 梅钦公式

1669 年，艾萨克·牛顿（Isaac Newton）和詹姆斯·格雷戈里（James Gregory）分别提出了三角函数 arcsin 和 arctan 的级数展开式，开启了解析方法计算 π 的新时代。π 求解方法最大的突破是找到快速收敛的反正切公式，这归功于 1706 年的英国数学家约翰·梅钦（John Machin）发现的梅钦公式。同年，威尔士数学家威廉·琼斯（William Jones）首先使用 π 符号作为圆的周长和直径的比值，该数学符号后来因 1737 年欧拉（Euler）在其著作中使用而被数学家广泛接纳沿用至今。约翰·梅钦也是第一个将圆周率手动计算突破 100 位小数大关的人。

梅钦公式以及后世梅钦类公式的原理如下，它们都是从角的和差公式中导出：对于两个角 α 和 β，它们角之和的正弦和余弦公式为：

$$\cos(\alpha+\beta)=\cos(\alpha)\cos(\beta)-\sin(\alpha)\sin(\beta)$$
$$\sin(\alpha+\beta)=\sin(\alpha)\cos(\beta)+\cos(\alpha)\sin(\beta)$$

其中当 $-\dfrac{\pi}{2}<\arctan(a_1/b_1)+\arctan(a_2/b_2)<\dfrac{\pi}{2}$ 时有如下方程成立：

$$\arctan(a_1/b_1)+\arctan(a_2/b_2)=\arctan\left(\frac{a_1b_2+a_2b_1}{b_1b_2-a_1a_2}\right)$$

比如，当 $a_1=a_2=1$，$b_1=b_2=5$ 时，有

$$2\arctan\left(\frac{1}{5}\right)=\arctan\left(\frac{1\times5+1\times5}{5\times5-1\times1}\right)=\arctan\left(\frac{5}{12}\right)$$

同理有 $4\arctan\left(\dfrac{1}{5}\right)=2\arctan\left(\dfrac{5}{12}\right)=\arctan\left(\dfrac{5\times12+5\times12}{12\times12-5\times5}\right)=\arctan\left(\dfrac{120}{119}\right)$

由于 $\arctan(1)=\dfrac{\pi}{4}$，则

$$4\arctan\left(\frac{1}{5}\right)-\frac{\pi}{4}=\arctan\left(\frac{120}{119}\right)-\arctan\left(\frac{1}{1}\right)=\arctan\left(\frac{120}{119}\right)+\arctan\left(\frac{-1}{1}\right)$$

即 $a_1=120$，$a_2=119$，$b_1=-1$，$b_2=1$ 代入公式得

$$4\arctan\left(\frac{1}{5}\right)-\frac{\pi}{4}=\arctan\left(\frac{120}{119}\right)+\arctan\left(\frac{-1}{1}\right)$$

$$=\arctan\left(\frac{20\times1+(-1)\times119}{119\times1-120\times(-1)}\right)=\arctan\left(\frac{1}{239}\right)$$

左右移位即得到梅钦公式如下：

$$\frac{\pi}{4}=4\arctan\left(\frac{1}{5}\right)-\arctan\left(\frac{1}{239}\right)$$

其中 $\arctan x$ 可由泰勒级数展开进行运算。基于梅钦公式英国数学家威廉·向克斯（William Shanks）花了近 15 年时间于 1873 年计算出圆周率小数点后 707 位。

由于梅钦公式并不是收敛最快的求解公式，数学家们还导出一系列与梅钦公式相似的公式，称为梅钦类公式。实际上所有的梅钦类公式主要是根据上面的方程反复构造所得，下面列出了若干梅钦类公式实例：

$$\frac{\pi}{4}=12\arctan\left(\frac{1}{18}\right)+8\arctan\left(\frac{1}{57}\right)-5\arctan\left(\frac{1}{239}\right)\quad\textbf{(Gauss)}$$

$$\frac{\pi}{4}=4\arctan\left(\frac{1}{5}\right)-\arctan\left(\frac{1}{70}\right)+\arctan\left(\frac{1}{99}\right)\quad\textbf{(Euler, E1)}$$

$$\frac{\pi}{4}=5\arctan\left(\frac{1}{7}\right)+2\arctan\left(\frac{3}{79}\right)\quad\textbf{(Euler, E2)}$$

$$\frac{\pi}{4}=\arctan\left(\frac{1}{2}\right)+\arctan\left(\frac{1}{3}\right)\quad\textbf{(Euler, E3)}$$

梅钦类公式是使用广泛的计算方法，总有不少数学家不断发现类似公式。比如 1982 年日本数学家兼诗人高野喜久雄（Kikuo Takano）发现的梅钦类公式为：

$$\frac{\pi}{4}=12\arctan\left(\frac{1}{49}\right)+32\arctan\left(\frac{1}{57}\right)-5\arctan\left(\frac{1}{239}\right)+12\arctan\left(\frac{1}{110443}\right)$$

对于威廉·向克斯计算出的707位结果，在70多年后数学家弗格森（D. F. Ferguson）对其中前608个数字的出现频率进行统计后发现0~9十个数字出现次数并不基本相同，于是他怀疑向克斯计算结果的正确性，并在1944年5月开始重新计算了一年多，于1946年确认威廉·向克斯的707位结果很遗憾地在第528位开始出错了。弗格森是用计算器计算圆周率小数点后位数精确到620位（1946年）的人，而1949年之前利用非计算机计算圆周率的最高纪录是1120位，之后就进入用计算机计算的时代。

4. 拉玛努金公式

1910年，最富直觉的印度天才数学家斯里尼瓦瑟·拉玛努金（Srinivasa Ramanujan）在梅钦类公式之外开辟了拉玛努金公式。他凭着对数字无与伦比的直觉探讨了π的几何构造近似值（3, 22/7, 355/113…）并发现众多π值计算方法，并于1910年发表了若干个快速收敛的无限级数，其中每计算一项都可获得8位数的有效精度。他的思维方法充满天才的直觉性，别人很难弄懂他是如何导出这些公式。他自称是从神那里得到这些可极大降低计算量的π值计算公式，其中一个与整数相关的简洁公式为

$$\frac{1}{\pi}=\frac{2\sqrt{2}}{9801}\sum_{k=0}^{\infty}\frac{(4k!)}{(k!)^4 4^{4k}}\frac{(1103+26390k)}{99^{4k}}=\frac{2\sqrt{2}}{9801}\sum_{k=0}^{\infty}\frac{(4k!)(1103+26390k)}{(k!)^4 396^{4k}}$$

5. 计算机求解时代

1949年，美国马里兰州的阿伯丁军队弹道研究实验室的数学家约翰·伦奇（John Wrench）等人用人类历史上的的第一台计算机（ENIAC）耗费70小时计算出了2037位小数，首次实现用计算机计算圆周率突破千位小数大关。圆周率的计算从此进入了计算机时代，之后由于计算机技术的发展，到1973年人类已经将圆周率的小数位计算推进到100万位。

1987年，美国数学家楚德诺夫斯基兄弟（David Chudnovsky & Gregory Chudnovsky）在拉玛努金公式的基础上改良出楚德诺夫斯基公式，该公式每计算一项可以得到15位的十进制精度。1994年，楚德诺夫斯基兄弟俩利用该公式算出圆周率小数点后40亿位数字，据说Wolfram Mathematica就是用该算法计算π值的。

$$\frac{1}{\pi}=12\sum_{k=0}^{\infty}(-1)^k\frac{(6k)!}{(3k)!(k!)^3}\frac{(545140134k+13591409)}{640320^{3k+\frac{3}{2}}}$$
$$=12\sum_{k=0}^{\infty}\frac{(-1)^k(6k)!(545140134k+13591409)}{(3k)!(k!)^3 640320^{3k+\frac{3}{2}}}$$

等价于：

$$\pi=\frac{426880\sqrt{10005}}{\sum_{k=0}^{\infty}\frac{(6k)!(545140134k+13591409)}{(3k)!(k!)^3(-640320)^{3k}}}$$

其中k值越大，则π值越精确。

6. 迭代计算方法

除了梅钦公式和拉玛努金公式之外，人们还找到迭代计算圆周率的算法。

（1）1975 年，尤金 • 萨拉明（Eugene Salamin）和理查德 • P. 布伦特（Richard Peirce Brent）各自独立发现了一个新的二次收敛迭代公式用于高精度圆周率快速计算，称为萨拉明—布伦特算法。该公式每经过一次计算，有效位数就会加倍。萨拉明—布伦特算法的原理如下：

设置初值 $x_0=1, y_0=1/\sqrt{2}, a_0=1/4, p_0=1$ 反复执行如下步骤直到 x_k 和 y_k 之间的误差小于指定阈值为止：

$$
\begin{aligned}
x_{k+1} &= (x_k + y_k)/2, \\
y_{k+1} &= \sqrt{x_k y_k}, \\
a_{k+1} &= a_k - p_k (x_k - x_{k+1})^2, \\
p_{k+1} &= 2p_k
\end{aligned}
$$

则最终求得近似值 $\pi \approx (x_{k+1}+y_{k+1})^2/(4a_{k+1})$，称为萨拉明—布伦特算法。

（2）后来人们发现卡尔 • 弗里德里希 • 高斯（1777—1855）以前也发现过类似但更复杂的公式，从而衍生出快速收敛的高斯—勒让德算法，其基本原理如下：

设置初值 $x_0=1, y_0=1/\sqrt{2}, a_0=1/2$ 反复执行如下步骤计算两个数的算术平均值和几何平均值，以逼近其算术—几何平均值：

$$
\begin{aligned}
x_{k+1} &= (x_k + y_k)/2 \\
y_{k+1} &= \sqrt{x_k y_k} \\
a_{k+1} &= a_k - 2^{k+1}({x_{k+1}}^2 - {y_{k+1}}^2)
\end{aligned}
$$

则当 $k\to\infty$ 时，有 $\pi = \lim\limits_{k\to\infty}(2{x_k}^2/a_k)$，称为高斯—勒让德二次型。

由于该算法每迭代一次可得双倍的十进制精度，计算百万位只需迭代 20 次即可。1999 年 9 月，日本的高桥大介和金田康正用高斯—勒让德公式计算了圆周率的 2061 亿位数字，创造了当时新的世界纪录。著名的计算机性能评估程序 Super PI 和 Hyper PI 就是使用此算法来评估计算机中央处理器在特定内存硬件条件下的运算性能。

（3）类似的迭代算法还有波尔文二次迭代式（Borwein Quadratic），其原理如下：

设置初值 $x_0=\sqrt{2}, y_0=0, a_0=2+\sqrt{2}$ 反复执行如下步骤：

$$
\begin{aligned}
x_{k+1} &= (\sqrt{x_k} + 1/\sqrt{x_k})/2 \\
y_{k+1} &= \sqrt{x_k}\left(\frac{1+y_k}{y_k+x_k}\right) \\
a_{k+1} &= a_k y_{k+1}\left(\frac{1+x_{k+1}}{1+y_{k+1}}\right)
\end{aligned}
$$

则当 $k\to\infty$ 时，$\pi = \lim\limits_{k\to\infty} a_k$，称为波尔文二次迭代式。

1985 年，彼得・波尔文（Peter Borwein）和乔纳森・波尔文（Jonathan Borwein）发现了收敛更快的波尔文四次迭代式。其原理如下：

设置初值 $y_0=\sqrt{2}-1, a_0=6-4\sqrt{2}$ 反复执行如下步骤：

$$y_{k+1}=\left(1-\sqrt[4]{(1-y_k^{\ 4})}\right)\Big/\left(1+\sqrt[4]{(1-y_k^{\ 4})}\right)$$

$$a_{k+1}=(1+y_k)^4a_k-2^{2k+3}y_{k+1}(1+y_{k+1}+y_{k+1}^{\ 2})$$

则当 $k\to\infty$ 时，$\pi=\lim\limits_{k\to\infty}1/a_k$，称为波尔文四次迭代式。

（4）1996 年 11 月，由大卫・贝利（David Bailey）、彼得・波尔文（Peter Borwein）和西蒙・普劳夫（Simon Plouffe）共同发表的贝利—波尔文—普劳夫公式（简称 BPP 公式）是划时代的圆周率算法，它能计算圆周率的任意第 n 位二进制或十六进制小数而不需要先计算前面的 n-1 位数字，算法复杂度为 $O(n^3\log(n)^3)$，该算法使得分布式并行计算圆周率成为可能。BBP 计算 π 的公式为

$$\pi=\sum_{k=0}^{\infty}\frac{1}{16^k}\left(\frac{4}{8k+1}-\frac{2}{8k+4}-\frac{1}{8k+5}-\frac{1}{8k+6}\right)$$

1997 年，法国天才程序员法布里斯・贝拉（Fabrice Bellard）发现了更加高效的 BBP 算法变体——贝拉公式（如下）。其计算复杂度为 $O(n^2)$，比西蒙・普劳夫在论文 *On the Computation of the n'th Decimal Digit of Various Transcendental Numbers* 中描述的算法更快约 43%。他曾在 2009 年末宣称其程序在 90 天内计算出了 π 的 2.69 万亿位小数，该算法至今依然是计算机求解 π 值最好的算法之一。

$$\pi=\frac{1}{2^6}\sum_{k=0}^{\infty}\frac{(-1)^k}{2^{10k}}\left(-\frac{2^5}{4k+1}-\frac{1}{4k+3}+\frac{2^8}{10k+1}-\frac{2^6}{10k+3}-\frac{2^2}{10k+5}-\frac{2^2}{10k+7}+\frac{1}{10k+9}\right)$$

2010 年以后大部分圆周率研究者都是基于余智恒（Alexander J. Yee）的 y-cruncher 多线程程序进行 π 值计算，2016 年彼得・楚伯（Peter Trueb）基于 y-Cruncher 计算出 π 的 22.46 万亿位小数（22,459,157,718,361），创造当时的最高纪录。

7. 积分和概率方法

除了前面的数学解析方法求解 π 值外，还可以借助计算机利用微积分方法和概率方法计算 π 值。其中前者借助三角函数在特定区间的面积计算来模拟，后者则根据特定分布规律的投点求积进行模拟，包括简单的蒙特卡罗方法和蒲丰投针方法。图 27-1 总结了圆周率的主要计算历史。

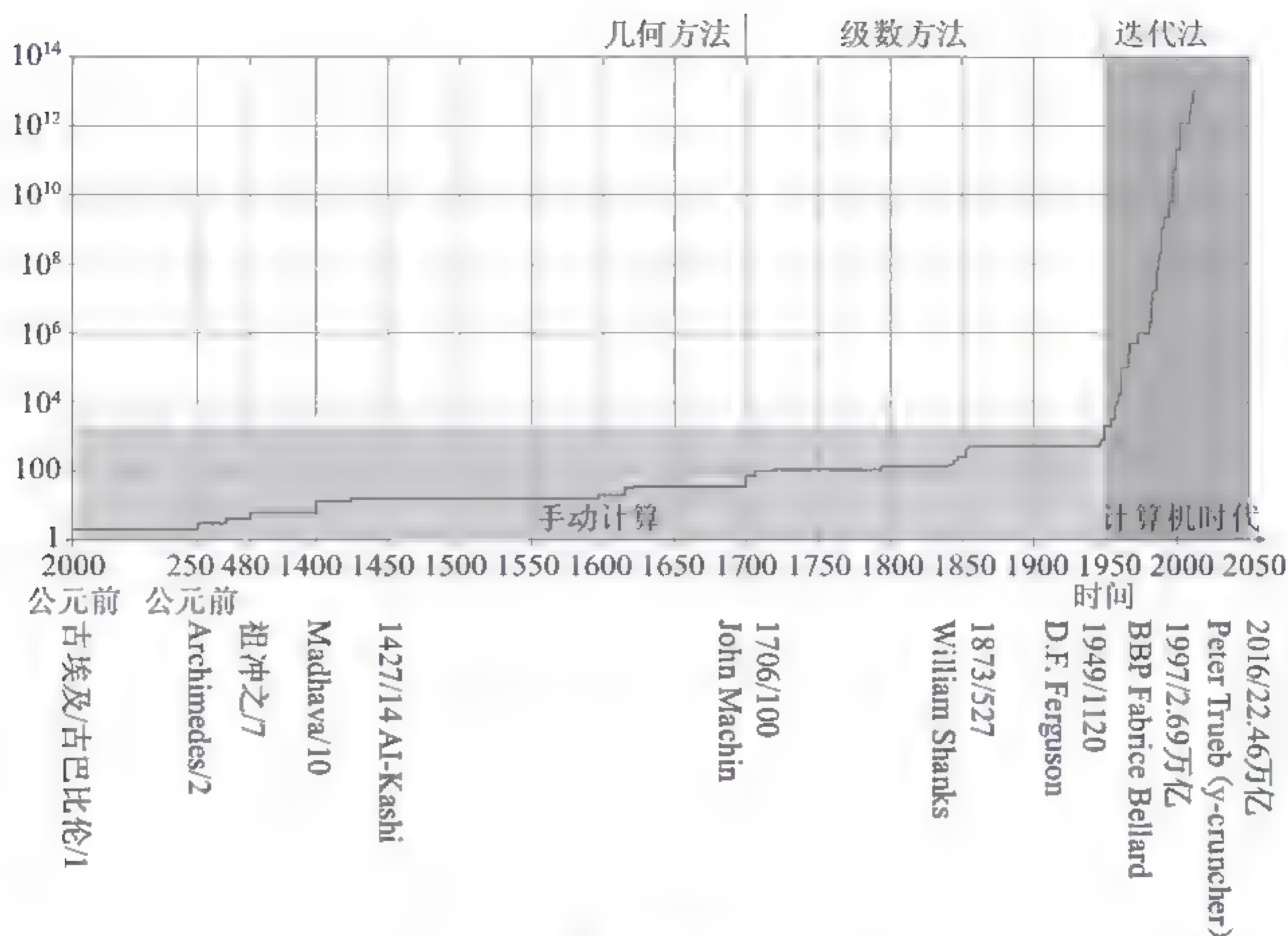


图 27-1　圆周率的计算史

下面从简单到复杂探索如何编写 SAS 程序计算 π 值，以及如何突破数据类型限制求解小数点后任意位数的精确值；最后探讨如何利用 SAS 对 π 值进行分析研究。

27.1.1　蒙特卡罗方法

如图 27-2 所示，从 π 的几何学定义可知，在边长为 1 的正方形内，如果投点遵从均匀分布，则投点落正方形内切四分之一圆内的个数在投点数量足够大时，就是其内切 1/4 圆形的面积 π/4。因此对于我们均匀分布的投点 (x, y)，如果满足 $x^2+y^2 \leqslant r^2$（其中 $r=1$）表示投点落在圆内，否则就落在圆外。因此，如果我们在平面上投掷的点数足够

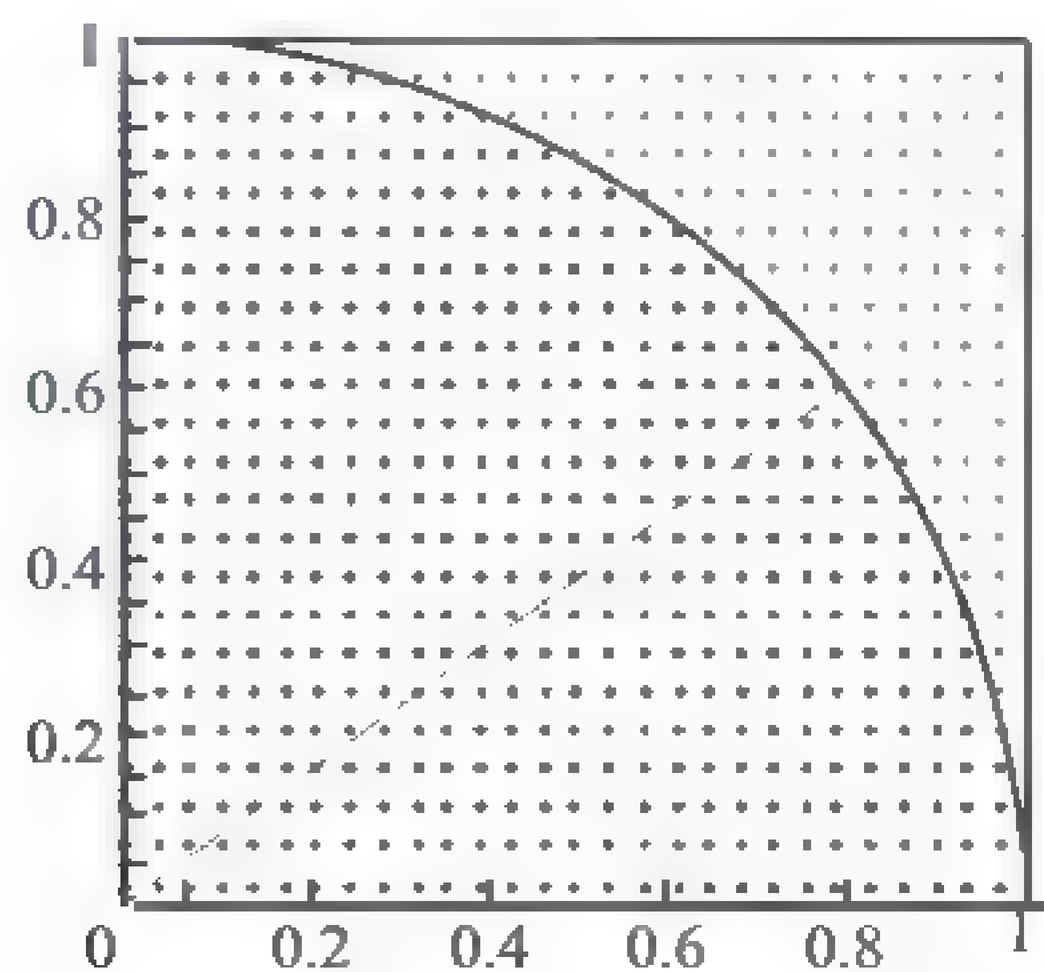


图 27-2　蒙特卡罗方法

多的话，我们将可计算出 1 4 圆的面积，由于我们取半径为 1，则其面积应该是 π/4。基于此思想我们可实现求 π 的 SAS 代码如程序 27-1 所示。

程序27-1　蒙特卡罗概率方法直观计算PI 值

```
data  null ;
  st=datetime();

  n=100000000; /*投掷 1亿次*/
  retain m  0;
  do i=1 to n;
    x=rand("UNIFORM");
    y=rand("UNIFORM"); /*要求满足均匀分布*/
    if x * x + y* y <= 1 then m=m+1;/*落在圆内的个数*/
  end;
  pi =4 * m / n;
  put pi= best.;

  et=datetime();
  pt=et-st;
  put "计算耗时：" pt time10.3" m/n: " m  "/" n  ;
run;
```

系统输出为 pi=3.14160516。

需要注意的一点是，由于是采用随机投点方法，要求随机数必须服从均匀分布而非正态分布，它在二维平面上要求具有等概率投点，否则上面的计算是无效的。基于这种投点法计算出的 π 值并不稳定，研究表明波动性较大，且小数点后的位数是有限的。

27.1.2　蒲丰投针方法

1777 年，法国数学家蒲丰（Buffon）发现的蒲丰投针问题，为利用计算机模拟来计算 π 值提供一种概率求解方法。蒲丰投针问题是说，在平面上画有间距为 $a(a>0)$ 的一系列平行线，然后向该平面上随机投掷长度为 $l(l<a)$ 的针，则蒲丰证明针与任一平行线相交的概率为 $2l/a\pi$，当针长 l 为 $a/2$ 时，其概率为 $1/\pi$。甚至可以证明长度为 l 的铁丝扔在平面上与平行线交点个数的期望值与铁丝的弯曲形状无关，而是与长度 l 成正比。比较直观的例子是将长度为 π 的铁丝弯曲成一个直径为 1 的正圆投掷在间距为 1 的平行线上总是有 2 个交点。

蒲丰投针思想的证明如下（见图 27-3）：假定长度为 l 的针中点为 M，它与最近的一条平行线的距离为 x，夹角为 y，则满足 $0 \leqslant x \leqslant a/2$ 且 $0 \leqslant y \leqslant \pi$。也就是说在右侧坐标 xy 平面中为了使得针与平行线有交点，中点到底线的距离必须小于等于针长度的一半乘以 $\sin(y)$，即 $x \leqslant l\ 2\sin(y)$。因此，它对应于右侧以 xy 为坐标轴的曲线下方的灰色区域。对于分布在 $a/2$ 和 π 空间中服从均匀分布的随机点，落入阴影部分的概率 p 为阴影面积 $\int_0^{\pi}\frac{1}{2}\sin(y)\,\mathrm{d}y$ 和矩形总面积 $\pi a/2$ 的比，其值等于 $2l/a\pi$ 。

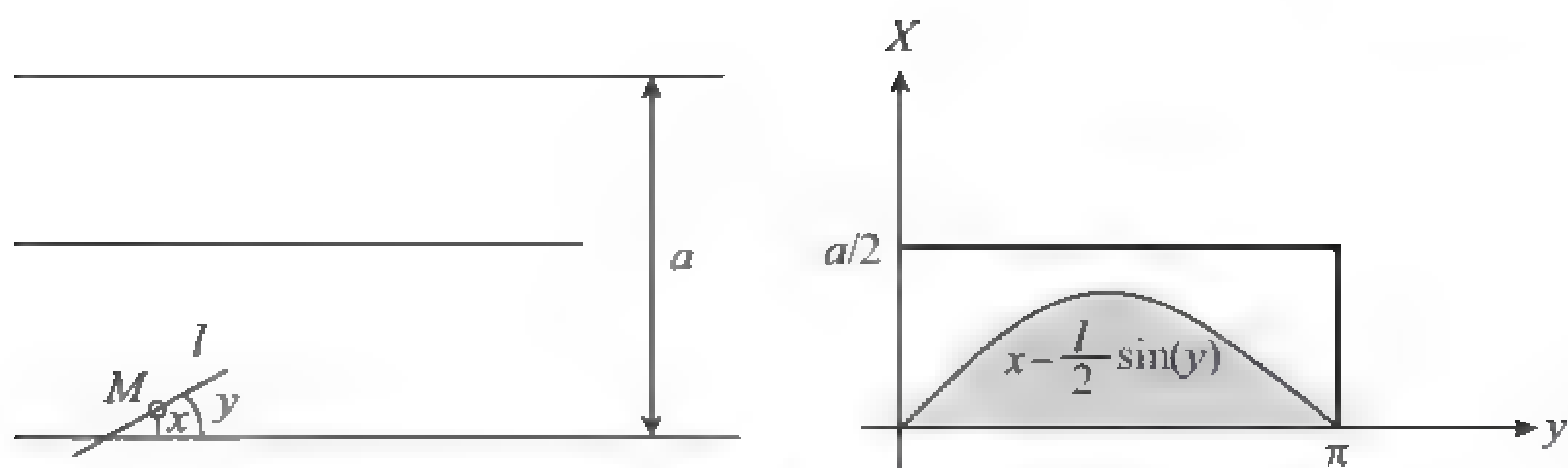


图 27-3　蒲丰投针方法

为简化计算，假定针的长度 $l=1$，平行线的间距 $a=2$ 来进行蒲丰投针试验，其完整 SAS 实现代码如程序 27-2 所示：

程序27-2　蒲丰投针概率方法计算PI 值

```
data _null_;
  st=datetime();

  n=10000000;
  retain m 0;

  d=2;
  do i=1 to n;
    x=rand("UNIFORM"); /* 0 - a/2 */
    y=rand("UNIFORM")* constant("PI"); /* 0 - π */

    if x<(1/2) * sin(y) then m=m+1;
  end;
  p= m/n;
  pi =1 / p;
  put pi= best.;

  et=datetime();
  pt=et-st;
  put "计算耗时: " pt time10.3" 命中次数/投针次数: " m  "/" n  ;
run;
```

系统输出为 pi=3.142160122

计算耗时 0:00:01.02 命中次数 / 投针次数： 3182524 /10000000

程序 27-2 输出显示蒲丰投针程序计算结果的精度很低，且由于该程序引用了系统预定义的常数 constant("PI") 和三角函数 sin(y)，而它们本身都与待求的 π 值有关，因此严格意义上讲该程序并非基于自然数的四则运算求得，只能作为一种理论上的验证程序，小数点后的精确位数依然非常少。

27.1.3　微积分方法

由于数学上已经证明反正切函数 $y=\arctan(x)$ 有 $dy=1/(1+x^2)dx$。而已知当 $x=1$ 时有 $\arctan(1)=\pi/4$，因此 π 就是函数 $f(x)=1/(1+x^2)$ 在区间 $x \in [0, 1]$ 面积的 4 倍（见图 27-4）。

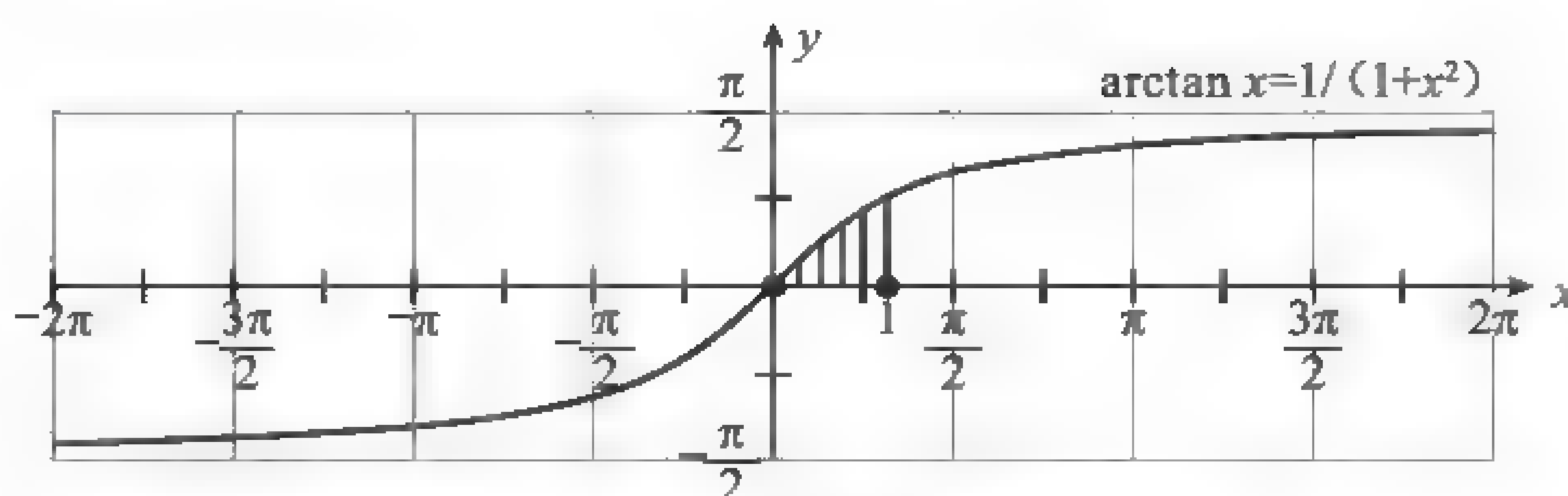


图 27-4　arctan 函数微积分方法

计算积分的思路就是将函数 $f(x)$ 和 x 轴之间的面积无限分割成许多小矩形单元，计算每个小矩形单元的面积求和。当分割得足够小时，该函数与 x 轴围成的面积将近无限接近 $\pi/4$。因此分割单元数是精度控制的关键，程序 27-3 将区间 $x \in [0, -1]$ 分成 1000 万个小单元实现 π 值计算。

程序27-3　函数微积分方法

```
data _null_;
  st=datetime();

  n=10000000; /*分割成一千万个小单元*/
  dx=1/n;
  x=0;
  y=0;
  do k=1 to n;
    x=x + dx;
    dy=1/ (1+ x * x );
    y=y+ dy;
  end;
  y= dx* y * 4;
  put y= best.;

  et=datetime();
  pt=et-st;
  put "计算耗时：" pt time10.2 ;
run;
```

系统输出如下，计算已经能够精确到小数点后 6 位数。

```
y=3.141592
计算耗时：0:00:00.08
```

27.1.4　幂级数方法

微积分方法固然不错，但如果能够把计算式表示为无穷级数序列，则可将计算精度提高一个层次。根据反正切函数的泰勒级数展开可知：

$$\arctan(x)=\sum_{k=0}^{\infty}(-1)^k \bullet \frac{1}{2k+1}x^{2k+1}=x-\frac{x^3}{3}+\frac{x^5}{5}+\cdots+(-1)^k \bullet x^{(2k+1)}/(2k+1)$$

当取特殊值时 $x=1$，有 $\arctan(1)=\pi/4$，则该公式建立了 π 和无穷级数序列之间的关系，此时得到如下公式，被称为 π 的莱布尼茨公式。

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots = \sum_{k=0}^{\infty}(-1)^k \cdot \frac{1}{2k+1}$$

其中 $k=0 \sim \infty$，$k+1$ 表示需要计算的项数。实际上，幂级数方法并不只是上面的一种方法，而是一序列类似机制的方法之总称。基于上面的公式我们很容易写出如下莱布尼茨幂级数实现程序 27-4 如下：

程序27-4　莱布尼茨幂级数方法

```
data _null_;
  b=1;   /*分母*/
  y=1;   /*求和*/
  sign=1;/*符号*/
  cnt=10000000;/*前面1千万项*/
  do k=0 to cnt-1;
    b=b+2;
    sign=sign* -1;
    y=y+ sign / b;
  end;
  y= y * 4;
  put y= best.;
run;
```

系统输出：y=3.1415927536，如果再额外计算一个级数项，则输出为 y=3.1415925536。

从上面计算可知，计算前面 10000000 项也只能算出小数点后 10 位数，并且由于奇偶数项的符号相反，收敛不是很快。当然除了上面的莱布尼茨级数公式，也可以使用其他幂级数公式来计算 π 值，比如下面的 π/2 递增级数计算公式：

$$\pi/2 = 1 + 1/5 \times 2/5 + 1/5 \times 2/5 \times 3/7 + \ldots$$

编程时除了从级数的项数进行控制外，也可以利用每一级数项的贡献大小进行控制，比如某项的贡献小于 10^{-15} 时即可结束计算（见程序 27-5）。可以看到系统输出跟上面程序输出稍有不同，但可看出 π 确实在 3.1415926 ~ 3.1415927，基本上达到了这种方法所能达到的精度极限。

程序27-5　单调递增的幂级数方法

```
data _null_;
  a=1; b=3;
  x=2;
  y=x;
  do while ( x>1E-15); /*用单项贡献控制精度*/
    x=x * a/b;
    y=y+x;
    a=a+1;
    b=b+2;
  end;
  put y= best.;
run;
```

系统输出：y=3.1415926536

27.1.5 幂级数高精度方法

前文尝试了蒙特卡罗方法、蒲丰投针方法、微积方法和幂级数方法计算π值，但所得结果的精度往往不超过小数点后10位数，部分算法计算不稳定且不精确，而导致不精确的原因不仅是采用计算方法的原因，还有在计算过程中使用了计算机系统中默认的浮点数存储表示。编程语言包括C/C++通常在内部采用32位或64位来表示单精度或双精度浮点数，由于数据存储的空间有限，精度损失在所难免。如果要获得高精度的π值计算，不但要使用无穷级数的方法，还要在计算过程中设计精度保持机制，其原理就是利用数组来存储部分十进制位，利用数组长度来控制所能计算的数值范围，最后基于这种存储结构来编写运算规则，包括基础的四则运算。

假定要计算π小数点后1000位数字，就要分配一个1000个元素的数组来存储数据的每一位数，每个元素对应于小数点后的每一位数字；然后将加法的进位，减法的借位以及除法的余数都在这些数组上进行，从而保留精度以免丢失。下面以单调递增的幂级数方法 $\frac{\pi}{2}=1+\frac{1}{5}+\frac{1}{5}\times\frac{2}{5}+\frac{1}{5}\times\frac{2}{5}\times\frac{3}{7}+\cdots$ 为例来实现（见程序27-6）。

程序27-6 单调递增的幂级数方法，增加精度保持特性

```
data _null_ ;
  st=datetime();

  L=1000;            /*小数点后位数*/

  L_MAX=L+1;         /*小数点后位数加上个位数共 1001 位*/
  array x[1001];     /*小数点前后总位数*/
  array y[1001];     /*小数点前后总位数*/

  a=1; b=3;
  do i=1 to L_MAX;
    x[i]=0;
    y[i]=0;
  end;
  x[1]=2;
  y[1]=x[1];

  N_MAX=constant("BIG"); /*最大迭代次数,可制定具体值来限定最大项数*/
  n=0; bContinue=1;
  do while( n< N_MAX & bContinue=1);
    /* 实现 x=x * a */
    carry=0;
    do i=L_MAX to 1 by -1;
      c=x[i] * a + carry;
      x[i]= mod(c, 10);
      carry = floor( c / 10);
    end;
    /* 实现 x=x / b */
    remain=0;
    do i=1 to L_MAX;
      c = x[i] + remain * 10;
      x[i] = floor(c /b);
      remain = mod(c, b);
    end;
    /* 实现 y=y + x */
```

```
    bContinue=0;
    do i=L_MAX to 2 by -1;
      c=y[i] + x[i];
      y[i]=mod(c, 10);
      carry=floor( c / 10);
      y[i-1] = y[i-1] + carry;
      if x[i]>0 then bContinue=1; /*不收敛则继续*/
    end;
    n=n+1;
    a=a+1; b=b+2;
  end;

  length s $ 32767; /*输出字符串长度*/
  s="";
  do i=1 to L_MAX;
    if i=2 then s=trim(left(s)) || ".";
    if mod(i-3,5)=4 then s=trim(left(s)) || " " ||  trim(left(y[i]));
    else s=trim(left(s)) || trim(left(y[i]));
  end;
  put "计算结果: " s;
  put "计算位数: " L;

  et=datetime();
  pt=et-st;
  if pt>0 then speed=n/pt;
  put "计算耗时: " pt time. " 迭代次数: " n " 计算速度: "  speed " 次/秒";
run;
```

运行上面的程序，系统将输出 π 小数点后 1000 位。改变 L 值可以输出指定长度的 π 的值。为显示和检查方便，下面的结果每行限定为 50 位数字。

计算位数 : 1000 计算耗时 :0:00:00 迭代次数 :3316 计算速度 : 9110 次 / 秒

计算结果：3.

```
14159 26535 89793 23846 26433 83279 50288 41971 69399 37510
58209 74944 59230 78164 06286 20899 86280 34825 34211 70679
82148 08651 32823 06647 09384 46095 50582 23172 53594 08128
48111 74502 84102 70193 85211 05559 64462 29489 54930 38196
44288 10975 66593 34461 28475 64823 37867 83165 27120 19091
45648 56692 34603 48610 45432 66482 13393 60726 02491 41273
72458 70066 06315 58817 48815 20920 96282 92540 91715 36436
78925 90360 01133 05305 48820 46652 13841 46951 94151 16094
33057 27036 57595 91953 09218 61173 81932 61179 31051 18548
07446 23799 62749 56735 18857 52724 89122 79381 83011 94912
98336 73362 44065 66430 86021 39494 63952 24737 19070 21798
60943 70277 05392 17176 29317 67523 84674 81846 76694 05132
00056 81271 45263 56082 77857 71342 75778 96091 73637 17872
14684 40901 22495 34301 46549 58537 10507 92279 68925 89235
42019 95611 21290 21960 86403 44181 59813 62977 47713 09960
51870 72113 49999 99837 29780 49951 05973 17328 16096 31859
50244 59455 34690 83026 42522 30825 33446 85035 26193 11881
71010 00313 78387 52886 58753 32083 81420 61717 76691 47303
59825 34904 28755 46873 11595 62863 88235 37875 93751 95778
18577 80532 17122 68066 13001 92787 66111 95909 21641 98681
```

至此，似乎到达了寻求精确 π 值的终点，然而上面的算法依然有瑕疵：①计算虽然能求出指定位数的 π 值，但检查发现最后 6 位（有下画线者）并不精确，因此一般要求多运算若干位并舍去期望位数后面的若干位来使得期望位数都是准确的；②该计算方法

并不是最快的，在提高运算效率上还可有进一步改进余地。该程序计算小数点后 1000 位的值，系统迭代了 3316 次，速度为 9110 次 / 秒。

27.1.6 梅钦类公式高精度方法

在 1706 年约翰·梅钦发现其公式之前，人们发现了一些简单的类似反正切函数表达式，如 π/4=arctan(1/2)+arctan(1/3)，而且 arctan(*x*) 已知可由如下幂级数展开公式计算得到

$$\arctan x = x - x^3/3 + x^5/5 + \cdots + (-1)^k x^{(2k+1)}/(2k+1)$$

其中 k=0 ~ ∞，k+1 表示需要计算的项数。因此计算值突破的方向就是如何找到简单且收敛快的公式，从前面的介绍已知梅钦公式是最早发现的快速公式：

$$\pi/4 = 4\arctan(1/5) - \arctan(1/239)$$

该公式结合前面已知 arctan(*x*) 可展开为无穷幂级数进行计算可知，可使用它来更快收敛。另外，也可从编程技巧上进行改进，如一个数组元素不是存储小数点后的一位数，而是若干位数，如果数组的一个元素存储 5 位数的话，梅钦公式中每计算一项则可以获得约 $Z\log_{10}(5)$=1.39794 位的十进制精度，其时间复杂度为 $O(n^2)$。

程序 27-7 为笔者基于梅钦公式结合大数求解技巧实现基础运算，包括加法、减法和除法。其中用来表示大数的数组，每个元素映射的是 5 位数字，而不像程序 27-5 的计算过程中是 1 位数字。

程序27-7 基于数组扩展基本运算规则，包括加法、减法和除法

```
proc fcmp outlib=work.funcs.pi;
  /*加法函数：单个数组元素表示5位数*/
  function add(a[*], b[*], c[*], m);
    outargs c;
    carry=0;
    do i=m to 1 by -1;
      c[i]=a[i]+ b[i]+ carry;
      if c[i]<100000 then carry=0;
      else do;
        c[i]=c[i]-100000;/*进位处理*/
        carry=1;
      end;
    end;
    return(1);
  endsub;
  /*减法函数：单个数组元素表示5位数*/
  function sub(a[*], b[*], c[*], m);
    outargs c;
    borrow=0;
    do i=m to 1 by -1;
      c[i]=a[i]- b[i]-borrow;
      if c[i]>=0 then borrow=0;
      else do;
        c[i]=c[i]+100000;/*借位处理*/
        borrow=1;
      end;
    end;
```

```
    return(1);
  endsub;
  /*除法函数：单个数组元素表示5位数*/
  function div(a[*], b, c[*], m);
    outargs c;
    remain=0;
    do i=1 to m;
      tmp=a[i] + remain;
      c[i] = floor(tmp / b); /*c[i] 是整数*/
      remain = mod(tmp, b ) * 100000;
    end;
    return(1);
  endsub;
run;
```

基于数组实现了基本运算法则后，就可以编写高精度梅钦公式求解程序（见程序 27-8）其中 *L* 表示需要计算的小数点后的总位数 5000 位。

程序27-8　Machin公式求解法，数组一个元素表示5位数

```
options cmplib=work.funcs;
data mydata;
  st=datetime();

  L=5000;  /* 需要计算的小数点后的总位数 */

  m=L/5+1; /* 如果每段 5 位数的话，只需 L/5+1 个区段=5000/5+1=1001*/
  array s[1001] _temporary_;
  array w[1001] _temporary_;
  array v[1001] _temporary_;
  array q[1001] _temporary_;
  do i=1 to 1001;
    s[i]=0;w[i]=0;v[i]=0;q[i]=0;
  end;

  /* PI/4= 4 arctan(1/5) -  arctan(1/239)
  => PI  =16 arctan(1/5) - 4 arctan(1/239)*/
  w[1] = 16 * 5;
  v[1] =  4 * 239;

  /*需要计算的项数，每计算一项可以得到 2log10[5]=1.39794 位的十进制精度*/
  n=floor( L / 1.39794 +1);

  do k= 0 to n;
    rc=div(w, 25, w, m);     /* 5   * 5 = 25    */
    rc=div(v, 57121,v, m); /* 239 * 239= 57121 */
    rc=sub(w, v, q, m);
    rc=div(q, 2*k+1, q, m);

    if mod(k,2)=0 then  rc=add (s,q,s, m); /* 奇数项 */
    else               rc=sub (s,q,s, m); /* 偶数项 */
  end;

  et=datetime();/*计时结束*/
  pt=et-st;
  if pt>0 then speed L/pt;
  put "计算耗时: " pt time10." 计算速度:"  speed 8.1" 次/秒";

  put s[1]; /*输出结果到日志*/
  do k=2 to m;
    if s[k]^=. then put s[k] z5." " @@;
```

```
    if mod(k-1, 10)=0 then put;
  end;
run;
```

系统输出如下所示，由于计算了更长的π值，注意它输出的994位到第1000位数字（下画线者）与前面的程序27-6是不同的，本方法输出的所有位数比前者更加精确，且计算速度更快。

计算耗时：0:00:00 计算速度 : 12285.0 次 / 秒

```
计算结果：3.
14159 26535 89793 23846 26433 83279 50288 41971 69399 37510
58209 74944 59230 78164 06286 20899 86280 34825 34211 70679
82148 08651 32823 06647 09384 46095 50582 23172 53594 08128
48111 74502 84102 70193 85211 05559 64462 29489 54930 38196
44288 10975 66593 34461 28475 64823 37867 83165 27120 19091
45648 56692 34603 48610 45432 66482 13393 60726 02491 41273
72458 70066 06315 58817 48815 20920 96282 92540 91715 36436
78925 90360 01133 05305 48820 46652 13841 46951 94151 16094
33057 27036 57595 91953 09218 61173 81932 61179 31051 18548
07446 23799 62749 56735 18857 52724 89122 79381 83011 94912
98336 73362 44065 66430 86021 39494 63952 24737 19070 21798
60943 70277 05392 17176 29317 67523 84674 81846 76694 05132
00056 81271 45263 56082 77857 71342 75778 96091 73637 17872
14684 40901 22495 34301 46549 58537 10507 92279 68925 89235
42019 95611 21290 21960 86403 44181 59813 62977 47713 09960
51870 72113 49999 99837 29780 49951 05973 17328 16096 31859
50244 59455 34690 83026 42522 30825 33446 85035 26193 11881
71010 00313 78387 52886 58753 32083 81420 61717 76691 47303
59825 34904 28755 46873 11595 62863 88235 37875 93751 95778
18577 80532 17122 68066 13001 92787 66111 95909 21642 01989
38095 25720 10654 85863 27886 59361 53381 82796 82303 01952
03530 18529 68995 77362 25994 13891 24972 17752 83479 13151
55748 57242 45415 06959 50829 53311 68617 27855 88907 50983
81754 63746 49393 19255 06040 09277 01671 13900 98488 24012
85836 16035 63707 66010 47101 81942 95559 61989 46767 83744
94482 55379 77472 68471 04047 53464 62080 46684 25906 94912
93313 67702 89891 52104 75216 20569 66024 05803 81501 93511
25338 24300 35587 64024 74964 73263 91419 92726 04269 92279
67823 54781 63600 93417 21641 21992 45863 15030 28618 29745
55706 74983 85054 94588 58692 69956 90927 21079 75093 02955
32116 53449 87202 75596 02364 80665 49911 98818 34797 75356
63698 07426 54252 78625 51818 41757 46728 90977 77279 38000
81647 06001 61452 49192 17321 72147 72350 14144 19735 68548
16136 11573 52552 13347 57418 49468 43852 33239 07394 14333
45477 62416 86251 89835 69485 56209 92192 22184 27255 02542
56887 67179 04946 01653 46680 49886 27232 79178 60857 84383
82796 79766 81454 10095 38837 86360 95068 00642 25125 20511
73929 84896 08412 84886 26945 60424 19652 85022 21066 11863
06744 27862 20391 94945 04712 37137 86960 95636 43719 17287
46776 46575 73962 41389 08658 32645 99581 33904 78027 59009
94657 64078 95126 94683 98352 59570 98258 22620 52248 94077
26719 47826 84826 01476 99090 26401 36394 43745 53050 68203
49625 24517 49399 65143 14298 09190 65925 09372 21696 46151
57098 58387 41059 78859 59772 97549 89301 61753 92846 81382
68683 86894 27741 55991 85592 52459 53959 43104 99725 24680
84598 72736 44695 84865 38367 36222 62609 91246 08051 24388
43904 51244 13654 97627 80797 71569 14359 97700 12961 60894
41694 86855 58484 06353 42207 22258 28488 64815 84560 28506
01684 27394 52267 46767 88952 52138 52254 99546 66727 82398
```

```
64565 96116 35488 62305 77456 49803 55936 34568 17432 41125
15076 06947 94510 96596 09402 52288 79710 89314 56691 36867
22874 89405 60101 50330 86179 28680 92087 47609 17824 93858
90097 14909 67598 52613 65549 78189 31297 84821 68299 89487
22658 80485 75640 14270 47755 51323 79641 45152 37462 34364
54285 84447 95265 86782 10511 41354 73573 95231 13427 16610
21359 69536 23144 29524 84937 18711 01457 65403 59027 99344
03742 00731 05785 39062 19838 74478 08478 48968 33214 45713
86875 19435 06430 21845 31910 48481 00537 06146 80674 91927
81911 97939 95206 14196 63428 75444 06437 45123 71819 21799
98391 01591 95618 14675 14269 12397 48940 90718 64942 31961
56794 52080 95146 55022 52316 03881 93014 20937 62137 85595
66389 37787 08303 90697 92077 34672 21825 62599 66150 14215
03068 03844 77345 49202 60541 46659 25201 49744 28507 32518
66600 21324 34088 19071 04863 31734 64965 14539 05796 26856
10055 08106 65879 69981 63574 73638 40525 71459 10289 70641
40110 97120 62804 39039 75951 56771 57700 42033 78699 36007
23055 87631 76359 42187 31251 47120 53292 81918 26186 12586
73215 79198 41484 88291 64470 60957 52706 95722 09175 67116
72291 09816 90915 28017 35067 12748 58322 28718 35209 35396
57251 21083 57915 13698 82091 44421 00675 10334 67110 31412
67111 36990 86585 16398 31501 97016 51511 68517 14376 57618
35155 65088 49099 89859 98238 73455 28331 63550 76479 18535
89322 61854 89632 13293 30898 57064 20467 52590 70915 48141
65498 59461 63718 02709 81994 30992 44889 57571 28289 05923
23326 09729 97120 84433 57326 54893 82391 19325 97463 66730
58360 41428 13883 03203 82490 37589 85243 74417 02913 27656
18093 77344 40307 07469 21120 19130 20330 38019 76211 01100
44929 32151 60842 44485 96376 69838 95228 68478 31235 52658
21314 49576 85726 24334 41893 03968 64262 43410 77322 69780
28073 18915 44110 10446 82325 27162 01052 65227 21116 60396
66557 30925 47110 55785 37634 66820 65310 98965 26918 62056
47693 12570 58635 66201 85581 00729 36065 98764 86117 91045
33488 50346 11365 76867 53249 44166 80396 26579 78771 85560
84552 96541 26654 08530 61434 44318 58676 97514 56614 06800
70023 78776 59134 40171 27494 70420 56223 05389 94561 31407
11270 00407 85473 32699 39081 45466 46458 80797 27082 66830
63432 85878 56983 05235 80893 30657 57406 79545 71637 75254
20211 49557 61581 40025 01262 28594 13021 64715 50979 25923
09907 96547 37612 55176 56751 35751 78296 66454 77917 45011
29961 48903 04639 94713 29621 07340 43751 89573 59614 58901
93897 13111 79042 97828 56475 03203 19869 15140 28708 08599
04801 09412 14722 13179 47647 77262 24142 54854 54033 21571
85306 14228 81375 85043 06332 17518 29798 66223 71721 59160
77166 92547 48738 98665 49494 50114 65406 28433 66393 79003
97692 65672 14638 53067 36096 57120 91807 63832 71664 16274
88880 07869 25602 90228 47210 40317 21186 08204 19000 42296
61711 96377 92133 75751 14959 50156 60496 31862 94726 54736
42523 08177 03675 15906 73502 35072 83540 56704 03867 43513
62222 47715 89150 49530 98444 89333 09634 08780 76932 59939
78054 19341 44737 74418 42631 29860 80998 88687 41326 04714
```

从结果可以看到 Machin 公式计算 5000 位耗时 1s 不到，迭代速度明显比前面程序 27-6 的计算速度更快。因此，很多现代的计算机求解高精度 π 值都是采用梅钦公式或其衍生公式进行求解。当然，扩展运算存储为数组来实现高精度计算，其性能跟底层数组长度有关系（进位借位问题），比如计算 50000 位小数跟计算 5000 位小数时可明显感觉到耗时变长。

27.1.7 迭代方法——贝拉公式

下面的代码为作者用SAS语言以FCMP函数实现的BBP算法变体——贝拉公式（见程序27-9）。其中扩展了3个求余函数和2个质数的有关函数，其中pow mod函数使用了第10章讲到的位运算操作函数bAnd和bRShift函数。主函数BBP Bellard返回圆周率小数点后指定位置*m*后的9位小数，其中9是在单次调用中保持精度可以获得的最大位数。

程序27-9 BBP算法变体——贝拉公式

```
proc fcmp outlib=work.funcs.bbp;
  /* 返回 (a*b) 乘积对 m 求余*/
  function mul_mod(a, b, m);
    return (mod( a * b, m));
  endsub;

  /* 返回 x 对 y 求余的逆*/
  function inv_mod( x,  y);
    u = x;
    v = y;
    c = 1;
    a = 0;

    do until (u=0);
      q =floor( v / u);

      t = c;
      c = a - q * c;
      a = t;

      t = u;
      u = v - q * u;
      v = t;
    end;

    a =floor(mod( a, y));
    if (a <0) then a = y + a;
    return( a);
  endsub;

  /* 返回(a^b) 对 m求余*/
  function pow_mod(  a,  b,   m);
    bb=b;

    r = 1;
    aa = a;
    do while (1);
      if bAnd(bb ,1)=1 then r = mul_mod(r, aa, m);
      bb = bRShift(bb ,1);
      if (bb = 0) then return(r);
      aa = mul_mod(aa, aa, m);
    end;
    return (r);
  endsub;

  /* 判断 n 是否为质数 */
  function is_prime(  n);
    if ( mod(n, 2) = 0) then return(0);
```

```
  r = floor ( sqrt(n) );
  do i = 3 to r by 2;
    if ( mod(n,  i) = 0) then return(0);
  end;
  return (1);
endsub;

/* 返回自然数n 下一个质数*/
function next_prime(  n);
  nn=n;
  do until( is_prime(nn) );
    nn=nn+1;
  end;
  return ( nn);
endsub;
/* 基于 BBP 变体贝拉公式计算 PI 的第 m 位十进制小数，为确保精度仅取9 位数字
 * 计算复杂度为 O(n^2) 比标准BBP 算法快43% */
function bbp_bellard(m);
  N = floor( ((m + 20) * log(10) / log(2)) );
  sum = 0.0;

  a = 3;
  do while (a<=(2 * N));
    vmax = floor(  (log(2 * N) / log(a)) );
    av = 1;
    do i = 0 to  vmax-1;
      av = av * a;
    end;

    s = 0;
    num = 1;
    den = 1;
    v = 0;
    kq = 1;
    kq2 = 1;

    do k = 1 to N;
      t = k;
      if (kq >= a) then do;
        do until ( mod(t , a) ^= 0) ;
          t = t / a;
          v=v-1;
        end;
        kq = 0;
      end;
      kq=kq+1;
      num = mul_mod(num, t, av);

      t = (2 * k - 1);
      if (kq2 >= a) then do;
        if (kq2 = a) then do;
          do until( mod(t, a) ^=0);
            t = t / a;
            v=v+1;
          end;
        end;
        kq2=kq2-a;
      end;
      den = mul_mod(den, t, av);
      kq2 = kq2 + 2;

      if (v >0) then do;
```

```
          t = inv_mod(den, av);
          t = mul_mod(t, num, av);
          t = mul_mod(t, k, av);
          do i = v to vmax-1;
            t = mul_mod(t, a, av);
          end;
          s=s +  t;
          if (s >= av) then s =s - av;
        end;
      end;
      t = pow_mod(10, m - 1, av);
      s = mul_mod(s, t, av);
      sum = mod(sum +   s / av, 1.0);

      a = next_prime(a);
    end;
    /*四字节最大整数为10位的2147483647，为确保精度仅取 9位数*/
    return( floor( sum * 1E9) );
  endsub;
run;
quit;
```

可以在 DATA 步中对该 FCMP 函数进行调用，如计算第 101 位开始的 9 位小数，可调如下程序 27-10。读者需要注意其中的输出格式被指定为 z9. 表示输出 9 位，如果位数不足的话前面自动补 0 显示。

程序27-10　获得指定位数 101 后的 9 位小数

```
options cmplib=work.funcs;
data _null_;
  x=bbp_bellard(101);
  put x= z9.;/*系统输出: x=821480865 ，计算第101 位不需先计算 0-100 位*/
run;
```

如果为了得到圆周率小数点后全部 100 位小数点，可通过如下程序 27-11 获得，注意其中输出的位数为 9 的整倍数。

程序27-11　输出全部 100位小数

```
options cmplib=work.funcs;
data _null_;
  n=100;
  do i=1 to n by 9;/*只需要每隔9 位计算即可*/
    d=bbp_bellard(i);
    put d z9. @@;
  end;
run;
```

为了让输出结果更加整齐便于检查，可以每 5 个数输出一个空格，见程序 27-12 所示。

程序27-12　格式化输出，每5个数输出一个空格

```
options cmplib=work.funcs;
data _null_;
  n=100;

  do i=1 to n by 5;
    x=substr(put(bbp_bellard(i), z9.),1,5);
    put x @@;
  end;
run;
```

本迭代方法的最大好处是可以求小数点后任意指定位置的数值，而不需要预先计算其前面的位数，这一性质使并行计算圆周率值成为可能，比如我们可以用 DS2 多线程分别计算，在所有计算结束后，在主线程中按照位置从前到后对结果进行顺序组装即可。

27.2　π 值分析

既然 π 值是唯一的数值序列，很好奇的一个问题就是它小数点后的数字服从什么规律分布呢？到目前为止人们根据经验认为它很可能是随机的，但由于常数 π 值后面的小数位是无穷无尽的，因此它并未得到有力的理论证明。为了研究分析这个问题，只能用计算机尽可能地计算出高精度数值，然后用数据分析的方法作经验性的证明。为了得到 π 的小数位序列供分析之用，以程序 27-8 为例进行改造。首先将小数点后的所有单个数字输出到 SAS 数据集作为观测进行考察。为此，将修改程序 27-8 中的如下代码片段（见程序 27-13）。

程序27-13　输出到日志窗口

```
put s[1]; /*输出结果到日志*/
do k=2 to m;
  if s[k]^=. then put s[k] z5." " @@;
  if mod(k-1, 10)=0 then put;
end;
```

上面的结果输出到日志窗口，而我们需要将计算出来的每 1 位数字输出到数据集 mydata 中，为此将代码更改为如程序 27-14 所示。

程序27-14　输出到SAS数据集

```
do k=2 to m;
  length x 8;
  str=put(s[k], z5.);
  do j=1 to 5;
    x=substr( str, j, 1);
    keep x;
    output;
  end;
end;
```

此时打印数据集 mydata 即可得到如图 27-5 所示的样本数据。

Obs	x		
1	1	4990	7
2	4	4991	4
3	1	4992	1
4	5	4993	3
5	9	4994	2
6	2	4995	6
7	6	4996	0
8	5	4997	4
9	3	4998	7
10	5	4999	1
		5000	4

图 27-5　将每 1 位数字作为观测输出

27.2.1 数字分布规律

首先使用 SGPLOT 过程步来了解其数字分布规律，运行如程序 27-15 可得到其频数分布直方图、核密度曲线以及参考的正态分布曲线。

程序27-15 考察单个位数的分布规律

```
proc sgplot data=mydata;
  title 'PI Digital Distribution';
  histogram x ;
  density x / type=normal legendlabel='Normal'
    lineattrs=(pattern=solid color=red);
  density x / type=kernel legendlabel='Kernel'
    lineattrs=(pattern=solid);
  keylegend / location=inside position=topright across=1;
  xaxis display=(nolabel) ;
run;
```

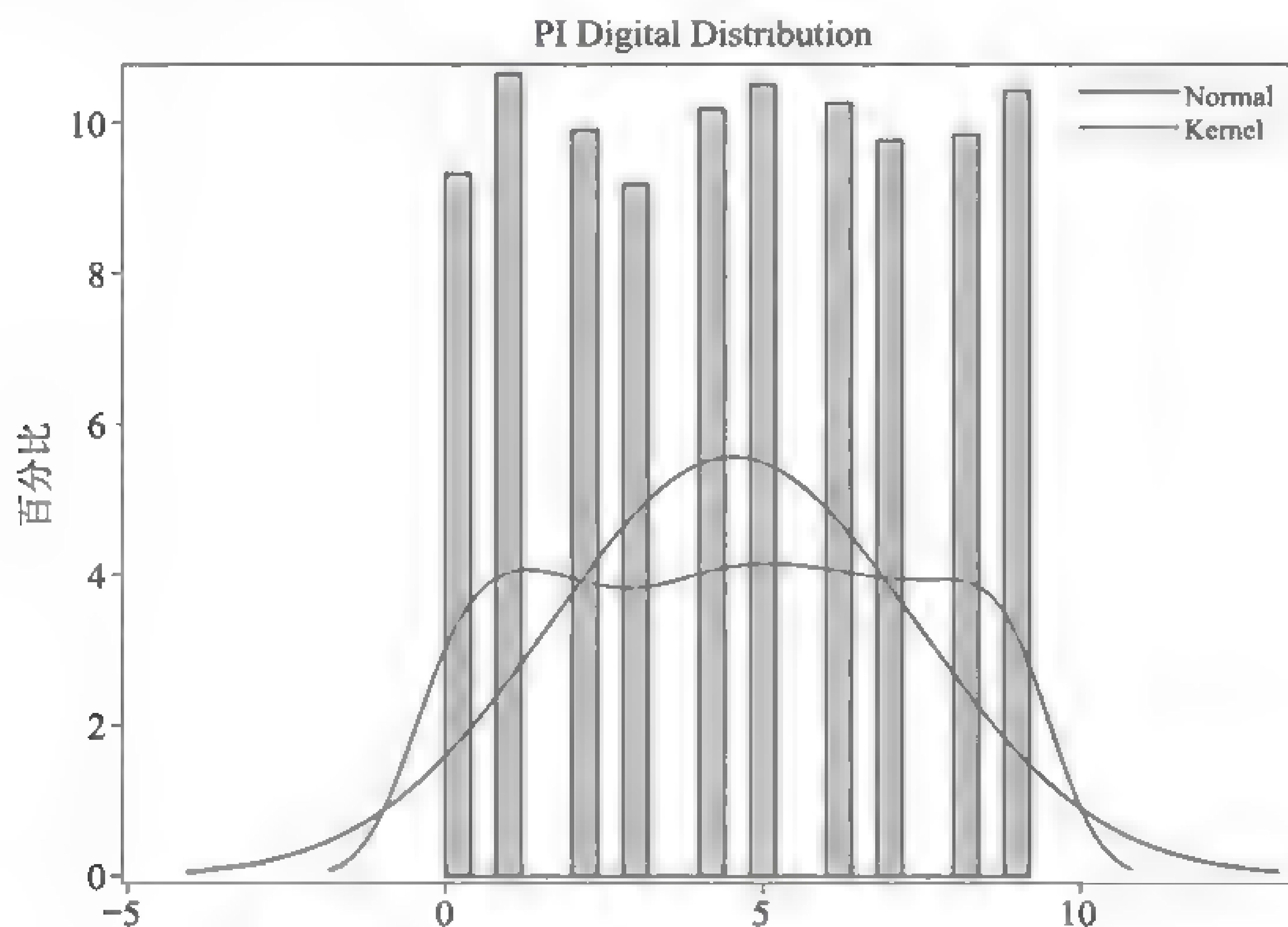


图 27-6 数字 0~9 的直方图和核密度曲线

从图 27-6 的概率密度分布曲线可知，它几近于等概分布（均匀分布）。随着求得的小数位数越多，其 0~9 共 10 个数字的频数基本一致。为更进一步分析其规律，可以分析数字出现次数和小数点位数的规律，即各个数字出现的总和及其平均值将会如何演变？生成 mydata_s 数据集，其中 *x*0~*x*9 分别表示数字 0~9 出现的总次数，*m*0~*m*9 表示各数字出现次数的平均值，即各数字出现的期望（见程序 27-16）。

程序27-16 计算数字出现次数以及平均出现次数与试验次数的关系

```
data mydata_s;
  set mydata;
  array a[0:9] x0-x9;
  array m[0:9] m0-m9;
  do i=0 to 9;
    if x=i then a[i]+1;
```

```
    m[i]= a[i]/_N_;
  end;
  t=_N_;
run;
```

一旦上面的程序生成了 mydata_s 数据集后，就可以绘制图形来揭示其规律。首先用程序 27-17 实现可重用的 SAS 宏，该宏将被用来绘制多条曲线。

程序27-17　绘制图形以揭示规律

```
%macro DrawPlot(prefix=x);
  proc sgplot data=mydata_s;
    title 'Mean/N Plot';
  %do i=0 %to 9;
    series x=t y=&prefix&i;
  %end;
  run;
%mend;
%DrawPlot(prefix=x);
%DrawPlot(prefix=m);
```

系统输出如下，如图 27-7 所示可以很清晰地看到，随着小数点位数的增长，各数字出现次数都近乎直线均匀增长，虽然其斜率稍有不同，但大致出现次数为总位数的 1/10。进一步地从图 27-8 所示也可以看出，所有数字的出现次数的期望都将收敛于 0.1 处，这说明从统计学上可以经验性地证明 π 的小数位的各个数字（0~9）服从均匀分布，即服从 p=1/10=0.1 的等概率离散均匀分布。

实际上，如果把 π 值小数点后每 m 位数作为考察对象，如 m=2 时，考察的对象就变成了 14 15 92 65 35 …此时不必更改求值程序，只需要基于前面生成的数据集重新分段构造出新数据集 mydata2 即可（见程序 27-18），然后将它作为数据样本进行分析，也可以得到类似的等概率分布规律（见图 27-9）。

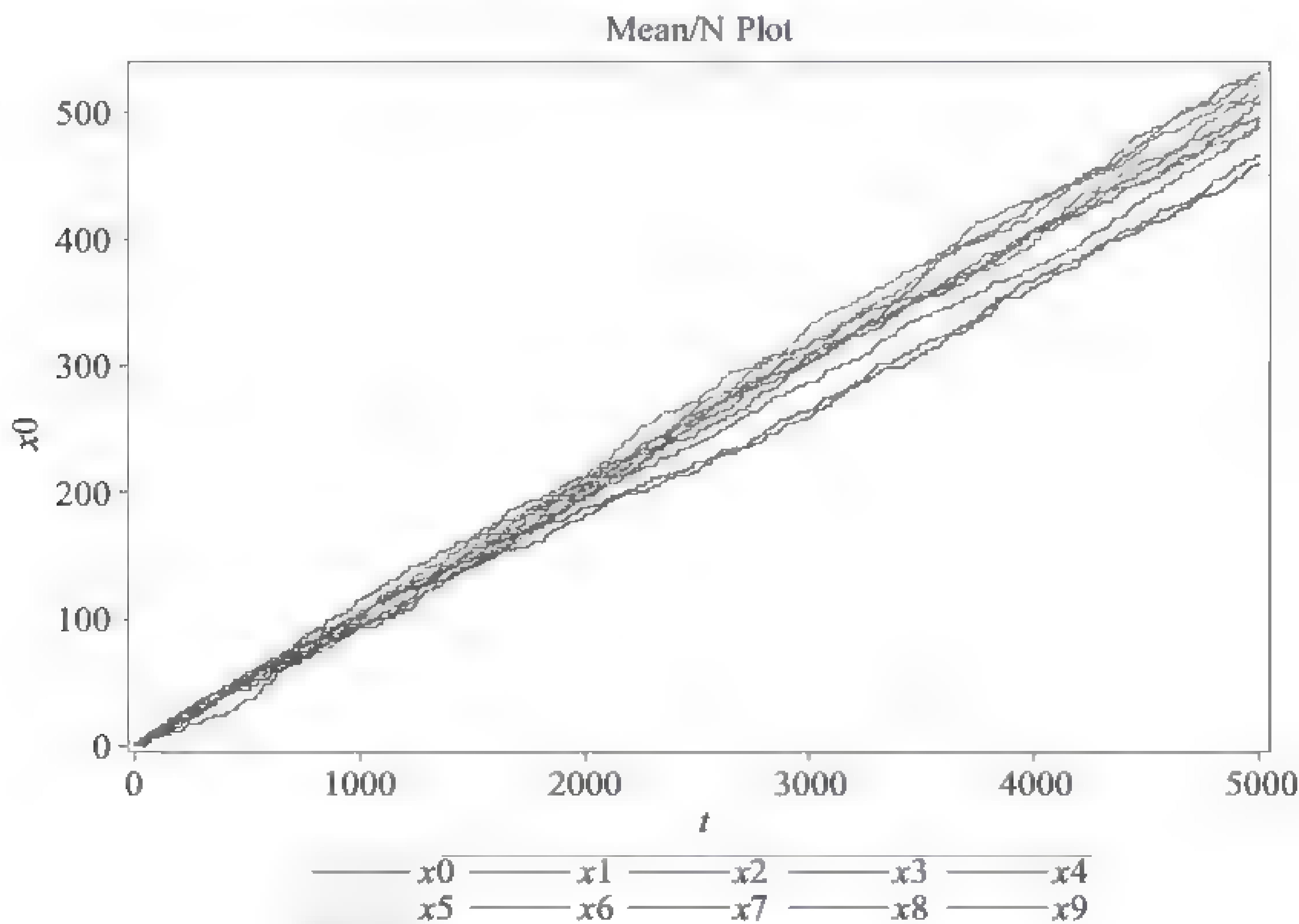


图 27-7　各条线具有大致相同的斜率 p=1/10

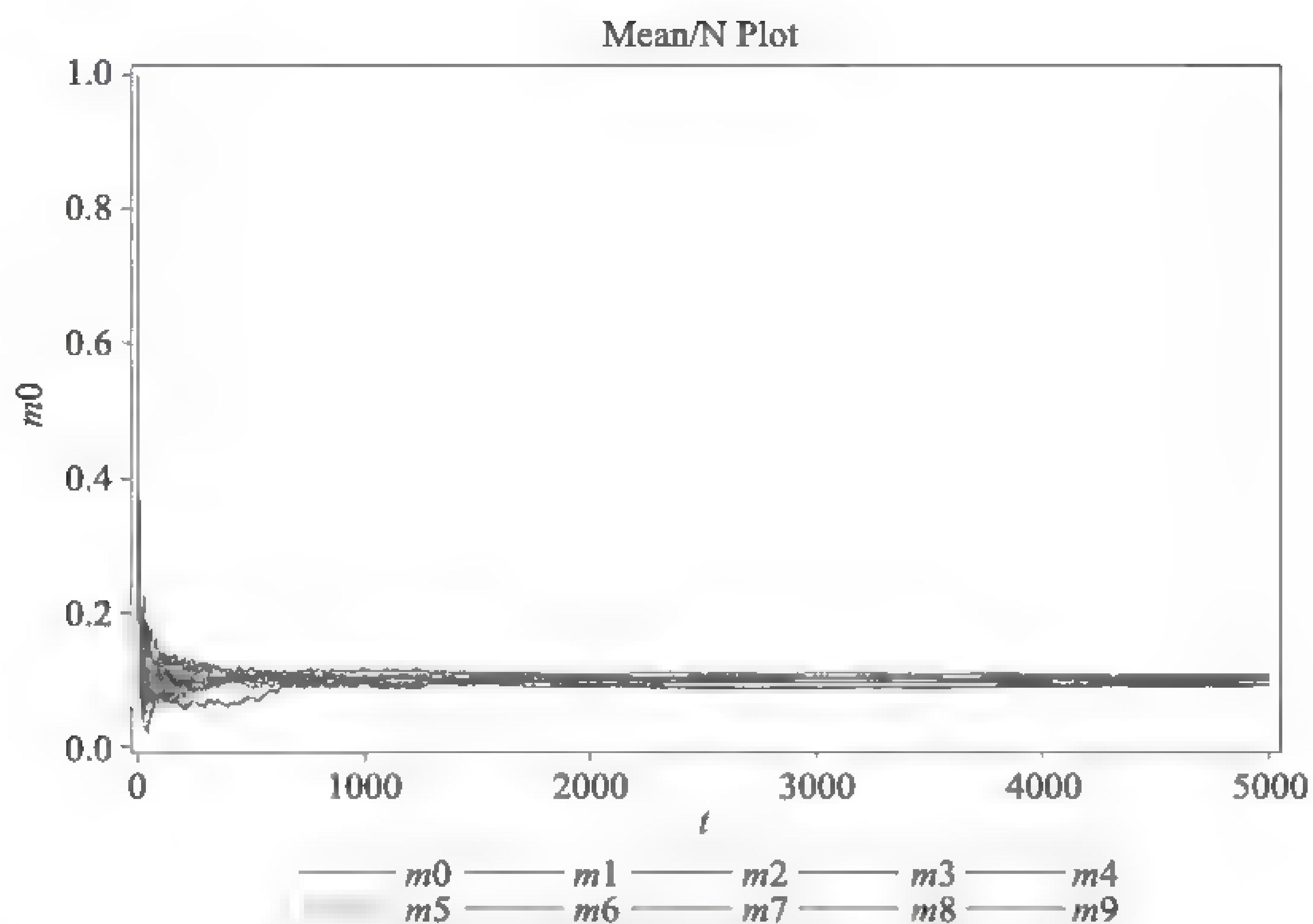


图 27-8　各数字的期望收敛于 0.1 处

程序27-18　将 m=2位小数位作为待考察的观测

```
data mydata2(rename=(sum=x));
  set mydata;
  retain sum;
  if mod(_N_,2)=1 then sum=x;/*每 2 位数重新构造 x 值*/
  else sum=sum*10+x;
  if mod(_N_,2)=0 then output;
  keep sum;
run;
```

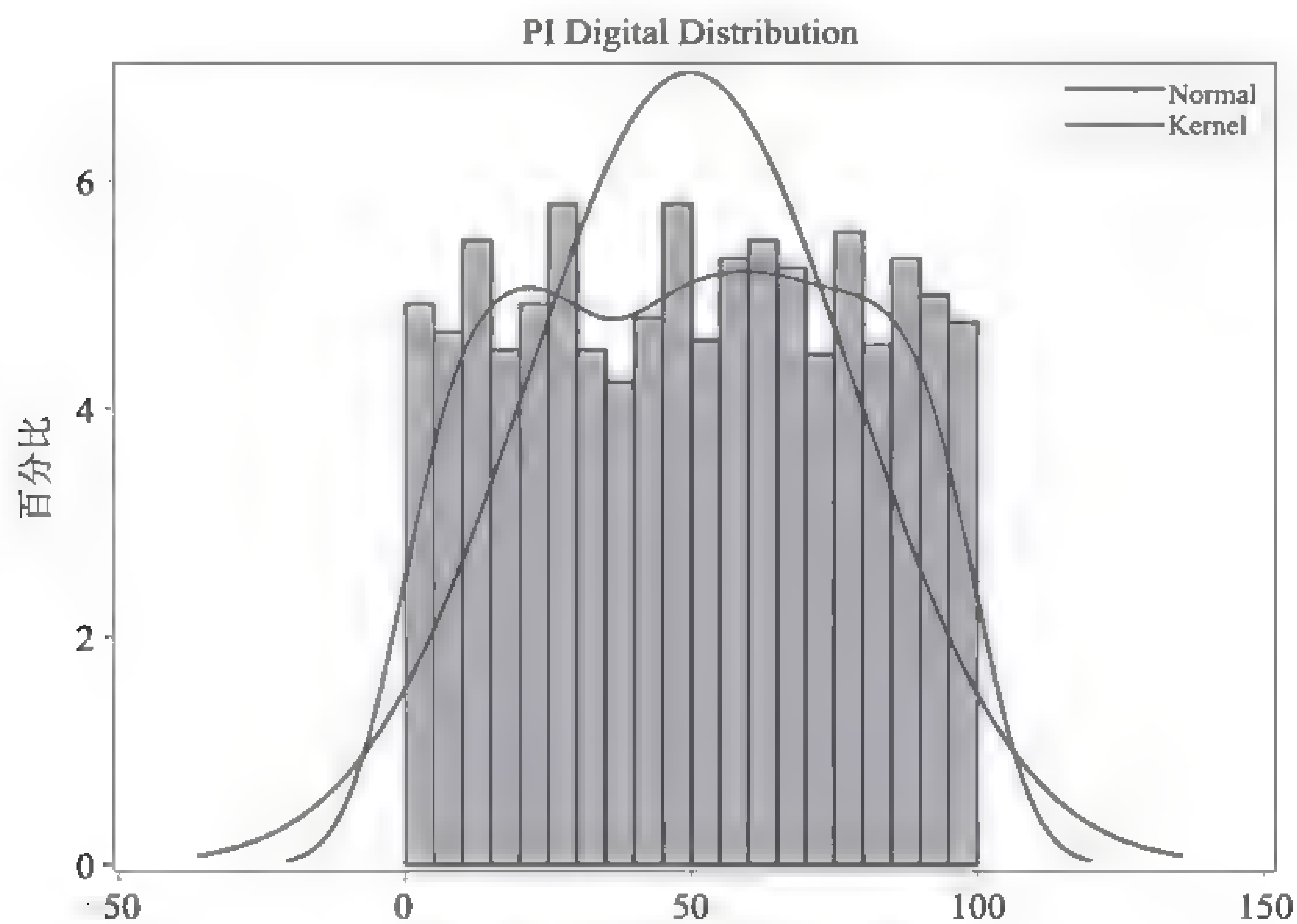


图 27-9　将两位数作为考察对象

27.2.2　可视化探索

更多的分析研究证明π的无限不循环小数序列不管是以每1个数字，还是以每2个数字或m个数字作为考察对象，该数字序列都服从等概率均匀分布（Uniform Distribution）特征。因此由于圆周率π的无限不循环特征，可以断言世界上任何长度的一串数字（比如任何人的出生日期）都可能在π的小数点中找到，因此圆周率π可能蕴含了关于这个世界的一切数字信息！π似乎在有序性和随机性之间找到了最为完美的平衡，但是关于它的随机性，尽管能够生成出有限长度的π值进行分析，目前尚无任何人能够从理论上证明其随机性。

1888年，英国数学家约翰•维恩（John Venn）曾手工绘制值π的前707个小数位的维恩图（也称文氏图），他将数字0~7分别指向顺时针方向排列的8个方向，同时忽略数字8和9，得到了如图27-10的图形模式，这些图像似乎暗示了π值随机性背后隐藏的固定模式。

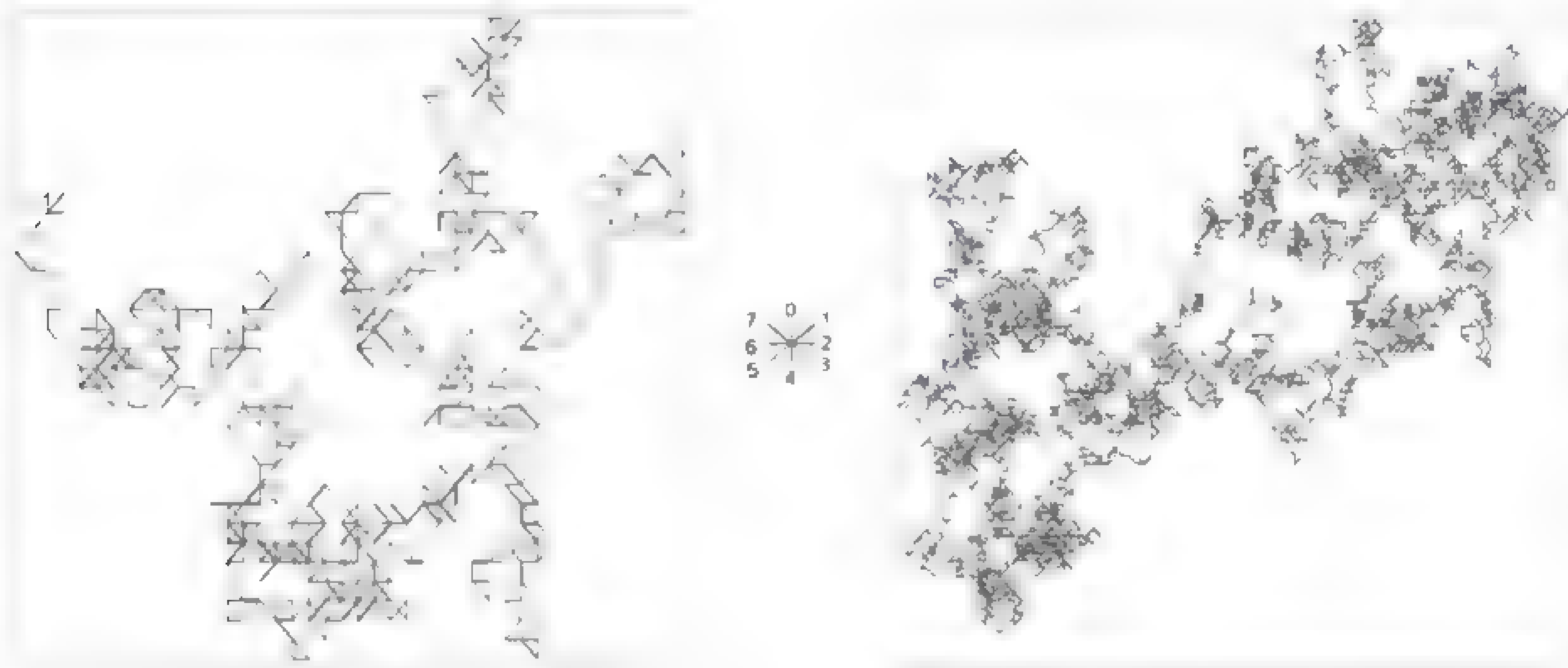


图 27-10　π 值前 707 和 4000 位数字生成的维恩图

笔者用SAS程序实现该图像的代码如程序27-19所示。

程序27-19　π的值彩色维恩图的绘制

```
data mydata_walk;
  set mydata;

  retain count 0;
  if count<=707; /*控制位数在0-7之间的项数*/
  if count=0 then do;
    yy=0;
    xx=0;
    output;
  end;

  if x=8 or x=9 then return; /*忽略 8 和 9*/

  if x=0 then do; yy=1; xx=0; end; /*根据数值控制偏移量*/
  if x=1 then do; yy=1; xx=1; end;
  if x=2 then do; yy=0; xx=1; end;
```

```
  if x=3 then do; yy=-1; xx= 1; end;
  if x=4 then do; yy=-1; xx= 0; end;
  if x=5 then do; yy=-1; xx=-1; end;
  if x=6 then do; yy= 0; xx=-1; end;
  if x=7 then do; yy= 1; xx=-1; end;

  retain cx cy 0;/*控制当前位置*/
  cx=cx+xx;
  cy=cy+yy;

  count=count+1;
  output;
run;
proc sgplot data=mydata_walk;
  series x=cx y=cy;
run;
```

得益于SAS强大的数据操纵和绘图能力，可以改进忽略掉数字8和9的维恩图，而是将数字0~9全部映射到顺时针方向排布的10个方向，每个方向之间夹角为36°（见程序27-20）。此时π值的维恩图可以绘制如下（见图27-11）。

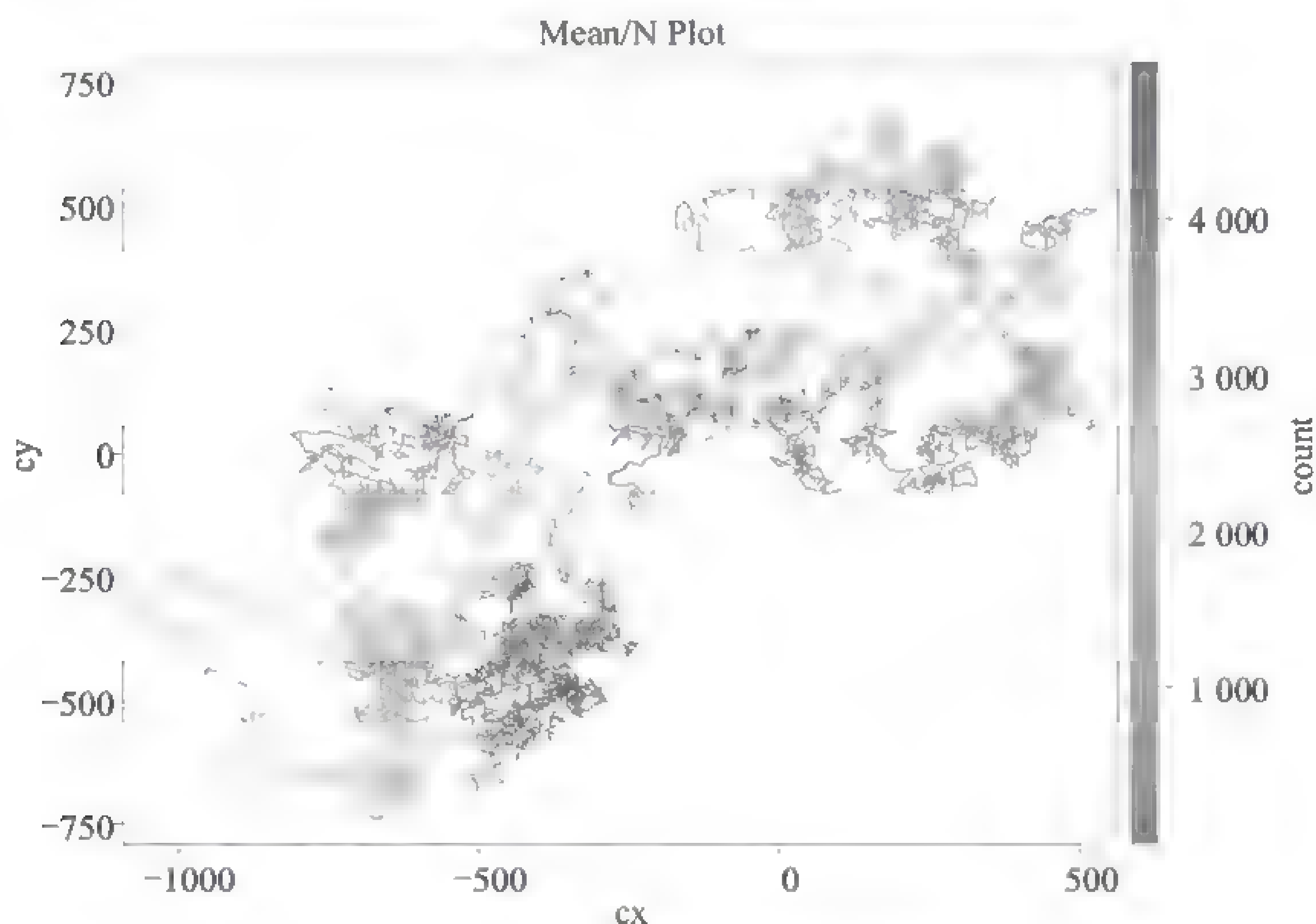


图27-11　5000位小数的π值维恩图

程序27-20　π值彩色维恩图的绘制

```
data mydata_walk;
  set mydata;

  if _N_=1 then do; yy=0; xx=0; output; end;

  theta= (x/10) * 2* 3.1415926536; /*择向*/
  yy= cos(theta);/*计算偏移量*/
  xx= sin(theta);

  retain cx cy 0;/*计算当前位置*/
```

```
  cx=cx+xx;
  cy=cy+yy;

  px=lag(cx);/*记录上一位置*/
  py=lag(cy);

  retain count 0;/*步进计数*/
  count=count+1;

  keep count cx cy px py;
  output;
run;
proc sgplot data=mydata_walk(where=(count<5001));
  vector x=cx y=cy /xorigin=px yorigin=py noarrowheads
    COLORRESPONSE=count colormodel=(green gold red);
run;
```

实际上，随着π值长度的增加，其数字背后隐藏的模式确实会呈现出某种确定性规律。图 27-12 显示了 10000 和 1000000 位 π 值的维恩图，其特定形状所揭示的规律跟分形图表现出某种相似性，但这些经验性特征至今尚未真正揭开 π 值包含一切数字神奇特性背后的理论真容。



图 27-12　10000 和 1000000 位值维恩图

回顾漫长的历史可以看到，人类在探寻 π 的道路上永不止步！曾经有这么个说法：一个国家能计算出的圆周率精确程度，可以衡量这个国家当时数学发展水平的指标。当然，现在借助计算机强大的计算能力，人类计算 π 值的限制更多地受限于计算机的计算能力。实际上，在日常工程计算中精确到小数点后 40 位的 π 值在天文学家计算银河系大小的圆周时，其误差已经小于一个质子。当下人类探索高精度 π 计算有个重要用途是用来衡量计算机软硬件系统的性能评估，以及探索该无限不循环数字中是否蕴含某些固定模式或规律。找到 π 后面无限不循环的数字只是事情的一个方面，而探索该数字内在的分析属性以及在某个领域的特殊作用则更具意义。因此关于 π 本身奥秘的探寻还远未结束，因为 π 依然还是那个永恒的、无尽变化且没有尽头的神秘常量，它包含一切可能的数字，背后依然还有太多太多的秘密等待我们去探索和分析！希望本章能够帮助读者揭开随机性世界的面纱一角，开启自己独立的数据分析和探索之路！

附　　录

附 录 1

二项分布累积概率表，查表参数为成功概率 p，独立伯努利试验次数 n 以及成功次数 m。

$$F_x(m)=P(X\leqslant m)=\sum_{x=0}^{m}\binom{n}{x}\cdot p^x\cdot q^{n-x}$$

程序28-1　生成二项分布的累积概率表

```
data ProbBNML;
  do n=1 to 20;
    do m=0 to (n-1);
      array c [19];
      i=1;
      do p=0.05 to 0.95 by 0.05;
        c[i]=ProbBNML(p,n,m); /*等价于 cdf( "BINOMIAL",m,p,n),p为成功概率*/
        i=i+1;
      end;
      keep n m c: ;
      format c: 9.4;
      output;
    end;
  end;
run;

/* 创建列标签 p=0.05 并打印输出*/
%macro SetLabel();
  data ProbBNML;
    set ProbBNML;
    %let i=1;
    %do i=1 %to 19;
      %let p= %sysfunc(putn(%sysevalf( &i * 0.05), 4.2));
      label c&i="&p";
    %end;
  run;
%mend;
%SetLabel();
proc print label;
run;
```

运行程序输出如下二项分布累积概率表。第 1 行为成功概率 p 的取值，第 1 列和第 2 列为独立伯努利试验次数 n 和成功次数 m 的取值（二项事件发生谓之成功），则对应单元格的值就是该参数 p，n 和 m 对应的累积概率（由于 $m=n$ 时累积概率恒为 1.0，表中不再列出）。

比如某药物的治愈率为 0.5，有 2 个人参加治疗，则无人治愈的概率为 0.25，无人治愈或者有 1 个人治愈的概率为 0.75，2 个人都治愈的概率为 1−0.75=0.25。

表 28-1 二项分布累积概率表

n	m	P																		
		0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90	0.95
1	0	0.9500	0.9000	0.8500	0.8000	0.7500	0.7000	0.6500	0.6000	0.5500	0.5000	0.4500	0.4000	0.3500	0.3000	0.2500	0.2000	0.1500	0.1000	0.0500
2	0	0.9025	0.8100	0.7225	0.6400	0.5625	0.4900	0.4225	0.3600	0.3025	0.2500	0.2025	0.1600	0.1225	0.0900	0.0625	0.0400	0.0225	0.0100	0.0025
2	1	0.9975	0.9900	0.9775	0.9600	0.9375	0.9100	0.8775	0.8400	0.7975	0.7500	0.6975	0.6400	0.5775	0.5100	0.4375	0.3600	0.2775	0.1900	0.0975
3	0	0.8574	0.7290	0.6141	0.5120	0.4219	0.3430	0.2746	0.2160	0.1664	0.1250	0.0911	0.0640	0.0429	0.0270	0.0156	0.0080	0.0034	0.0010	0.0001
3	1	0.9928	0.9720	0.9393	0.8960	0.8438	0.7840	0.7183	0.6480	0.5748	0.5000	0.4253	0.3520	0.2818	0.2160	0.1563	0.1040	0.0608	0.0280	0.0073
3	2	0.9999	0.9990	0.9966	0.9920	0.9844	0.9730	0.9571	0.9360	0.9089	0.8750	0.8336	0.7840	0.7254	0.6570	0.5781	0.4880	0.3859	0.2710	0.1426
4	0	0.8145	0.6561	0.5220	0.4096	0.3164	0.2401	0.1785	0.1296	0.0915	0.0625	0.0410	0.0256	0.0150	0.0081	0.0039	0.0016	0.0005	0.0001	0.0000
4	1	0.9860	0.9477	0.8905	0.8192	0.7383	0.6517	0.5630	0.4752	0.3910	0.3125	0.2415	0.1792	0.1265	0.0837	0.0508	0.0272	0.0120	0.0037	0.0005
4	2	0.9995	0.9963	0.9880	0.9728	0.9492	0.9163	0.8735	0.8208	0.7585	0.6875	0.6090	0.5248	0.4370	0.3483	0.2617	0.1808	0.1095	0.0523	0.0140
4	3	1.0000	0.9999	0.9995	0.9984	0.9961	0.9919	0.9850	0.9744	0.9590	0.9375	0.9085	0.8704	0.8215	0.7599	0.6836	0.5904	0.4780	0.3439	0.1855
5	0	0.7738	0.5905	0.4437	0.3277	0.2373	0.1681	0.1160	0.0778	0.0503	0.0313	0.0185	0.0102	0.0053	0.0024	0.0010	0.0003	0.0001	0.0000	0.0000
5	1	0.9774	0.9185	0.8352	0.7373	0.6328	0.5282	0.4284	0.3370	0.2562	0.1875	0.1312	0.0870	0.0540	0.0308	0.0156	0.0067	0.0022	0.0005	0.0000
5	2	0.9988	0.9914	0.9734	0.9421	0.8965	0.8369	0.7648	0.6826	0.5931	0.5000	0.4069	0.3174	0.2352	0.1631	0.1035	0.0579	0.0266	0.0086	0.0012
5	3	1.0000	0.9995	0.9978	0.9933	0.9844	0.9692	0.9460	0.9130	0.8688	0.8125	0.7438	0.6630	0.5716	0.4718	0.3672	0.2627	0.1648	0.0815	0.0226
5	4	1.0000	1.0000	0.9999	0.9997	0.9990	0.9976	0.9948	0.9898	0.9816	0.9688	0.9497	0.9222	0.8840	0.8319	0.7627	0.6723	0.5563	0.4095	0.2262
6	0	0.7351	0.5314	0.3772	0.2621	0.1780	0.1177	0.0754	0.0467	0.0277	0.0156	0.0083	0.0041	0.0018	0.0007	0.0002	0.0001	0.0000	0.0000	0.0000
6	1	0.9672	0.8857	0.7765	0.6554	0.5339	0.4202	0.3191	0.2333	0.1636	0.1094	0.0692	0.0410	0.0223	0.0109	0.0046	0.0016	0.0004	0.0001	0.0000
6	2	0.9978	0.9842	0.9527	0.9011	0.8306	0.7443	0.6471	0.5443	0.4415	0.3438	0.2553	0.1792	0.1174	0.0705	0.0376	0.0170	0.0059	0.0013	0.0001
6	3	0.9999	0.9987	0.9941	0.9830	0.9624	0.9295	0.8826	0.8208	0.7447	0.6563	0.5585	0.4557	0.3529	0.2557	0.1694	0.0989	0.0473	0.0159	0.0022
6	4	1.0000	1.0000	0.9996	0.9984	0.9954	0.9891	0.9777	0.9590	0.9308	0.8906	0.8364	0.7667	0.6809	0.5798	0.4661	0.3446	0.2235	0.1143	0.0328
6	5	1.0000	1.0000	1.0000	0.9999	0.9998	0.9993	0.9982	0.9959	0.9917	0.9844	0.9723	0.9533	0.9246	0.8824	0.8220	0.7379	0.6229	0.4686	0.2649
7	0	0.6983	0.4783	0.3206	0.2097	0.1335	0.0824	0.0490	0.0280	0.0152	0.0078	0.0037	0.0016	0.0006	0.0002	0.0001	0.0000	0.0000	0.0000	0.0000
7	1	0.9556	0.8503	0.7166	0.5767	0.4450	0.3294	0.2338	0.1586	0.1024	0.0625	0.0357	0.0188	0.0090	0.0038	0.0013	0.0004	0.0001	0.0000	0.0000
7	2	0.9962	0.9743	0.9262	0.8520	0.7564	0.6471	0.5323	0.4199	0.3164	0.2266	0.1529	0.0963	0.0556	0.0288	0.0129	0.0047	0.0012	0.0002	0.0000
7	3	0.9998	0.9973	0.9879	0.9667	0.9294	0.8740	0.8002	0.7102	0.6083	0.5000	0.3917	0.2898	0.1999	0.1260	0.0706	0.0333	0.0121	0.0027	0.0002
7	4	1.0000	0.9998	0.9988	0.9953	0.9871	0.9712	0.9444	0.9037	0.8471	0.7734	0.6836	0.5801	0.4677	0.3529	0.2436	0.1480	0.0738	0.0257	0.0038
7	5	1.0000	1.0000	0.9999	0.9996	0.9987	0.9962	0.9910	0.9812	0.9643	0.9375	0.8976	0.8414	0.7662	0.6706	0.5551	0.4233	0.2834	0.1497	0.0444
7	6	1.0000	1.0000	1.0000	1.0000	0.9999	0.9998	0.9994	0.9984	0.9963	0.9922	0.9848	0.9720	0.9510	0.9177	0.8665	0.7903	0.6794	0.5217	0.3017
8	0	0.6634	0.4305	0.2725	0.1678	0.1001	0.0577	0.0319	0.0168	0.0084	0.0039	0.0017	0.0007	0.0002	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000

（续表）

n	m	P																		
		0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90	0.95
8	1	0.9428	0.8131	0.6572	0.5033	0.3671	0.2553	0.1691	0.1064	0.0632	0.0352	0.0181	0.0085	0.0036	0.0013	0.0004	0.0001	0.0000	0.0000	0.0000
8	2	0.9942	0.9619	0.8948	0.7969	0.6785	0.5518	0.4278	0.3154	0.2201	0.1445	0.0885	0.0498	0.0253	0.0113	0.0042	0.0012	0.0002	0.0000	0.0000
8	3	0.9996	0.9950	0.9787	0.9437	0.8862	0.8059	0.7064	0.5941	0.4770	0.3633	0.2604	0.1737	0.1061	0.0580	0.0273	0.0104	0.0029	0.0004	0.0000
8	4	1.0000	0.9996	0.9972	0.9896	0.9727	0.9420	0.8939	0.8263	0.7396	0.6367	0.5230	0.4059	0.2936	0.1941	0.1138	0.0563	0.0214	0.0050	0.0004
8	5	1.0000	1.0000	0.9998	0.9988	0.9958	0.9887	0.9747	0.9502	0.9115	0.8555	0.7799	0.6846	0.5722	0.4482	0.3215	0.2031	0.1052	0.0381	0.0058
8	6	1.0000	1.0000	1.0000	0.9999	0.9996	0.9987	0.9964	0.9915	0.9819	0.9648	0.9368	0.8936	0.8309	0.7447	0.6329	0.4967	0.3428	0.1869	0.0572
8	7	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9998	0.9993	0.9983	0.9961	0.9916	0.9832	0.9681	0.9424	0.8999	0.8322	0.7275	0.5695	0.3366
9	0	0.6303	0.3874	0.2316	0.1342	0.0751	0.0404	0.0207	0.0101	0.0046	0.0020	0.0008	0.0003	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
9	1	0.9288	0.7748	0.5995	0.4362	0.3003	0.1960	0.1211	0.0705	0.0385	0.0195	0.0091	0.0038	0.0014	0.0004	0.0001	0.0000	0.0000	0.0000	0.0000
9	2	0.9916	0.9470	0.8592	0.7382	0.6007	0.4628	0.3373	0.2318	0.1495	0.0898	0.0498	0.0250	0.0112	0.0043	0.0013	0.0003	0.0001	0.0000	0.0000
9	3	0.9994	0.9917	0.9661	0.9144	0.8343	0.7297	0.6089	0.4826	0.3614	0.2539	0.1658	0.0994	0.0536	0.0253	0.0100	0.0031	0.0006	0.0001	0.0000
9	4	1.0000	0.9991	0.9944	0.9804	0.9511	0.9012	0.8283	0.7334	0.6214	0.5000	0.3786	0.2666	0.1717	0.0988	0.0489	0.0196	0.0056	0.0009	0.0000
9	5	1.0000	0.9999	0.9994	0.9969	0.9900	0.9747	0.9464	0.9007	0.8342	0.7461	0.6386	0.5174	0.3911	0.2703	0.1657	0.0856	0.0339	0.0083	0.0006
9	6	1.0000	1.0000	1.0000	0.9997	0.9987	0.9957	0.9888	0.9750	0.9502	0.9102	0.8505	0.7682	0.6627	0.5372	0.3993	0.2618	0.1409	0.0530	0.0084
9	7	1.0000	1.0000	1.0000	1.0000	0.9999	0.9996	0.9986	0.9962	0.9909	0.9805	0.9615	0.9295	0.8789	0.8040	0.6997	0.5638	0.4005	0.2252	0.0712
9	8	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9997	0.9992	0.9981	0.9954	0.9899	0.9793	0.9597	0.9249	0.8658	0.7684	0.6126	0.3698
10	0	0.5987	0.3487	0.1969	0.1074	0.0563	0.0283	0.0135	0.0061	0.0025	0.0010	0.0003	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
10	1	0.9139	0.7361	0.5443	0.3758	0.2440	0.1493	0.0860	0.0464	0.0233	0.0107	0.0045	0.0017	0.0005	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000
10	2	0.9885	0.9298	0.8202	0.6778	0.5256	0.3828	0.2616	0.1673	0.0996	0.0547	0.0274	0.0123	0.0048	0.0016	0.0004	0.0001	0.0000	0.0000	0.0000
10	3	0.9990	0.9872	0.9500	0.8791	0.7759	0.6496	0.5138	0.3823	0.2660	0.1719	0.1020	0.0548	0.0260	0.0106	0.0035	0.0009	0.0001	0.0000	0.0000
10	4	0.9999	0.9984	0.9901	0.9672	0.9219	0.8497	0.7515	0.6331	0.5044	0.3770	0.2616	0.1662	0.0949	0.0474	0.0197	0.0064	0.0014	0.0002	0.0000
10	5	1.0000	0.9999	0.9986	0.9936	0.9803	0.9527	0.9051	0.8338	0.7384	0.6231	0.4956	0.3669	0.2485	0.1503	0.0781	0.0328	0.0099	0.0016	0.0001
10	6	1.0000	1.0000	0.9999	0.9991	0.9965	0.9894	0.9740	0.9452	0.8980	0.8281	0.7340	0.6177	0.4862	0.3504	0.2241	0.1209	0.0500	0.0128	0.0010
10	7	1.0000	1.0000	1.0000	0.9999	0.9996	0.9984	0.9952	0.9877	0.9726	0.9453	0.9004	0.8327	0.7384	0.6172	0.4744	0.3222	0.1798	0.0702	0.0115
10	8	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9995	0.9983	0.9955	0.9893	0.9767	0.9536	0.9141	0.8507	0.7560	0.6242	0.4557	0.2639	0.0861
10	9	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9997	0.9990	0.9975	0.9940	0.9865	0.9718	0.9437	0.8926	0.8031	0.6513	0.4013
11	0	0.5688	0.3138	0.1673	0.0859	0.0422	0.0198	0.0088	0.0036	0.0014	0.0005	0.0002	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

（续表）

n	m	P																		
		0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90	0.95
11	1	0.8981	0.6974	0.4922	0.3221	0.1971	0.1130	0.0606	0.0302	0.0139	0.0059	0.0022	0.0007	0.0002	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000
11	2	0.9848	0.9104	0.7788	0.6174	0.4552	0.3127	0.2001	0.1189	0.0652	0.0327	0.0148	0.0059	0.0020	0.0006	0.0001	0.0000	0.0000	0.0000	0.0000
11	3	0.9985	0.9815	0.9306	0.8389	0.7133	0.5696	0.4256	0.2963	0.1911	0.1133	0.0610	0.0293	0.0122	0.0043	0.0012	0.0002	0.0000	0.0000	0.0000
11	4	0.9999	0.9973	0.9841	0.9496	0.8854	0.7897	0.6683	0.5328	0.3971	0.2744	0.1738	0.0994	0.0501	0.0216	0.0076	0.0020	0.0003	0.0000	0.0000
11	5	1.0000	0.9997	0.9973	0.9884	0.9657	0.9218	0.8513	0.7535	0.6331	0.5000	0.3669	0.2465	0.1487	0.0782	0.0343	0.0117	0.0027	0.0003	0.0000
11	6	1.0000	1.0000	0.9997	0.9980	0.9924	0.9784	0.9499	0.9007	0.8262	0.7256	0.6029	0.4672	0.3317	0.2103	0.1146	0.0504	0.0159	0.0028	0.0001
11	7	1.0000	1.0000	1.0000	0.9998	0.9988	0.9957	0.9878	0.9707	0.9390	0.8867	0.8089	0.7037	0.5745	0.4304	0.2867	0.1611	0.0694	0.0185	0.0016
11	8	1.0000	1.0000	1.0000	1.0000	0.9999	0.9994	0.9980	0.9941	0.9852	0.9673	0.9348	0.8811	0.7999	0.6873	0.5448	0.3826	0.2212	0.0896	0.0152
11	9	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9998	0.9993	0.9978	0.9941	0.9861	0.9698	0.9394	0.8870	0.8029	0.6779	0.5078	0.3026	0.1019
11	10	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9995	0.9986	0.9964	0.9913	0.9802	0.9578	0.9141	0.8327	0.6862	0.4312
12	0	0.5404	0.2824	0.1422	0.0687	0.0317	0.0138	0.0057	0.0022	0.0008	0.0002	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
12	1	0.8816	0.6590	0.4435	0.2749	0.1584	0.0850	0.0424	0.0196	0.0083	0.0032	0.0011	0.0003	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
12	2	0.9804	0.8891	0.7358	0.5584	0.3907	0.2528	0.1513	0.0834	0.0421	0.0193	0.0079	0.0028	0.0009	0.0002	0.0000	0.0000	0.0000	0.0000	0.0000
12	3	0.9978	0.9744	0.9078	0.7946	0.6488	0.4925	0.3467	0.2253	0.1345	0.0730	0.0356	0.0153	0.0056	0.0017	0.0004	0.0001	0.0000	0.0000	0.0000
12	4	0.9998	0.9957	0.9761	0.9274	0.8424	0.7237	0.5834	0.4382	0.3044	0.1939	0.1117	0.0573	0.0255	0.0095	0.0028	0.0006	0.0001	0.0000	0.0000
12	5	1.0000	0.9995	0.9954	0.9806	0.9456	0.8822	0.7873	0.6652	0.5269	0.3872	0.2607	0.1582	0.0846	0.0386	0.0143	0.0039	0.0007	0.0001	0.0000
12	6	1.0000	1.0000	0.9993	0.9961	0.9858	0.9614	0.9154	0.8418	0.7393	0.6128	0.4731	0.3348	0.2127	0.1179	0.0544	0.0194	0.0046	0.0005	0.0000
12	7	1.0000	1.0000	0.9999	0.9994	0.9972	0.9905	0.9745	0.9427	0.8883	0.8062	0.6956	0.5618	0.4167	0.2763	0.1576	0.0726	0.0239	0.0043	0.0002
12	8	1.0000	1.0000	1.0000	0.9999	0.9996	0.9983	0.9944	0.9847	0.9644	0.9270	0.8655	0.7747	0.6534	0.5075	0.3512	0.2054	0.0922	0.0256	0.0022
12	9	1.0000	1.0000	1.0000	1.0000	1.0000	0.9998	0.9992	0.9972	0.9921	0.9807	0.9579	0.9166	0.8487	0.7472	0.6093	0.4417	0.2642	0.1109	0.0196
12	10	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9997	0.9989	0.9968	0.9917	0.9804	0.9576	0.9150	0.8416	0.7251	0.5565	0.3410	0.1184
12	11	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9998	0.9992	0.9978	0.9943	0.9862	0.9683	0.9313	0.8578	0.7176	0.4596
13	0	0.5133	0.2542	0.1209	0.0550	0.0238	0.0097	0.0037	0.0013	0.0004	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

（续表）

n	m	P																		
		0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90	0.95
13	1	0.8646	0.6213	0.3983	0.2337	0.1267	0.0637	0.0296	0.0126	0.0049	0.0017	0.0005	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
13	2	0.9755	0.8661	0.6920	0.5017	0.3326	0.2025	0.1132	0.0579	0.0269	0.0112	0.0041	0.0013	0.0004	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000
13	3	0.9969	0.9658	0.8820	0.7473	0.5843	0.4206	0.2783	0.1686	0.0929	0.0461	0.0203	0.0078	0.0025	0.0007	0.0001	0.0000	0.0000	0.0000	0.0000
13	4	0.9997	0.9935	0.9658	0.9009	0.7940	0.6543	0.5005	0.3530	0.2280	0.1334	0.0699	0.0321	0.0126	0.0040	0.0010	0.0002	0.0000	0.0000	0.0000
13	5	1.0000	0.9991	0.9925	0.9700	0.9198	0.8346	0.7159	0.5744	0.4268	0.2905	0.1788	0.0977	0.0462	0.0182	0.0057	0.0013	0.0002	0.0000	0.0000
13	6	1.0000	0.9999	0.9987	0.9930	0.9757	0.9376	0.8705	0.7712	0.6437	0.5000	0.3563	0.2288	0.1295	0.0624	0.0243	0.0070	0.0013	0.0001	0.0000
13	7	1.0000	1.0000	0.9998	0.9988	0.9944	0.9818	0.9538	0.9023	0.8212	0.7095	0.5732	0.4256	0.2841	0.1654	0.0802	0.0300	0.0075	0.0009	0.0000
13	8	1.0000	1.0000	1.0000	0.9998	0.9990	0.9960	0.9874	0.9679	0.9302	0.8666	0.7721	0.6470	0.4995	0.3457	0.2060	0.0991	0.0342	0.0065	0.0003
13	9	1.0000	1.0000	1.0000	1.0000	0.9999	0.9994	0.9975	0.9922	0.9797	0.9539	0.9071	0.8314	0.7217	0.5794	0.4158	0.2527	0.1180	0.0342	0.0031
13	10	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9997	0.9987	0.9959	0.9888	0.9731	0.9421	0.8868	0.7975	0.6674	0.4984	0.3080	0.1339	0.0245
13	11	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9995	0.9983	0.9951	0.9874	0.9704	0.9363	0.8733	0.7664	0.6017	0.3787	0.1354
13	12	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9996	0.9987	0.9963	0.9903	0.9762	0.9450	0.8791	0.7458	0.4867
14	0	0.4877	0.2288	0.1028	0.0440	0.0178	0.0068	0.0024	0.0008	0.0002	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
14	1	0.8470	0.5846	0.3567	0.1979	0.1010	0.0475	0.0205	0.0081	0.0029	0.0009	0.0003	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
14	2	0.9700	0.8416	0.6479	0.4481	0.2811	0.1608	0.0839	0.0398	0.0170	0.0065	0.0022	0.0006	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
14	3	0.9958	0.9559	0.8535	0.6982	0.5213	0.3552	0.2205	0.1243	0.0632	0.0287	0.0114	0.0039	0.0011	0.0003	0.0000	0.0000	0.0000	0.0000	0.0000
14	4	0.9996	0.9908	0.9533	0.8702	0.7415	0.5842	0.4227	0.2793	0.1672	0.0898	0.0426	0.0175	0.0060	0.0017	0.0003	0.0001	0.0000	0.0000	0.0000
14	5	1.0000	0.9985	0.9885	0.9562	0.8883	0.7805	0.6405	0.4859	0.3373	0.2120	0.1189	0.0583	0.0243	0.0083	0.0022	0.0004	0.0000	0.0000	0.0000
14	6	1.0000	0.9998	0.9978	0.9884	0.9617	0.9067	0.8164	0.6925	0.5461	0.3953	0.2586	0.1501	0.0753	0.0315	0.0103	0.0024	0.0003	0.0000	0.0000
14	7	1.0000	1.0000	0.9997	0.9976	0.9897	0.9685	0.9247	0.8499	0.7414	0.6047	0.4539	0.3076	0.1836	0.0933	0.0383	0.0116	0.0022	0.0002	0.0000
14	8	1.0000	1.0000	1.0000	0.9996	0.9979	0.9917	0.9757	0.9417	0.8811	0.7880	0.6627	0.5142	0.3595	0.2195	0.1117	0.0439	0.0115	0.0015	0.0000
14	9	1.0000	1.0000	1.0000	1.0000	0.9997	0.9983	0.9940	0.9825	0.9574	0.9102	0.8328	0.7207	0.5773	0.4158	0.2585	0.1298	0.0467	0.0092	0.0004
14	10	1.0000	1.0000	1.0000	1.0000	1.0000	0.9998	0.9989	0.9961	0.9886	0.9713	0.9368	0.8757	0.7795	0.6448	0.4787	0.3018	0.1465	0.0441	0.0042
14	11	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9994	0.9979	0.9935	0.9830	0.9602	0.9161	0.8392	0.7189	0.5520	0.3521	0.1584	0.0301
14	12	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9998	0.9991	0.9971	0.9919	0.9795	0.9525	0.8990	0.8021	0.6433	0.4154	0.1530
14	13	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9998	0.9992	0.9976	0.9932	0.9822	0.9560	0.8972	0.7712	0.5123
15	0	0.4633	0.2059	0.0874	0.0352	0.0134	0.0048	0.0016	0.0005	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

（续表）

n	m	P																		
		0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90	0.95
15	1	0.8291	0.5490	0.3186	0.1671	0.0802	0.0353	0.0142	0.0052	0.0017	0.0005	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
15	2	0.9638	0.8159	0.6042	0.3980	0.2361	0.1268	0.0617	0.0271	0.0107	0.0037	0.0011	0.0003	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
15	3	0.9945	0.9444	0.8227	0.6482	0.4613	0.2969	0.1727	0.0905	0.0424	0.0176	0.0063	0.0019	0.0005	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000
15	4	0.9994	0.9873	0.9383	0.8358	0.6865	0.5155	0.3519	0.2173	0.1204	0.0592	0.0255	0.0094	0.0028	0.0007	0.0001	0.0000	0.0000	0.0000	0.0000
15	5	1.0000	0.9978	0.9832	0.9390	0.8516	0.7216	0.5643	0.4032	0.2608	0.1509	0.0769	0.0338	0.0124	0.0037	0.0008	0.0001	0.0000	0.0000	0.0000
15	6	1.0000	0.9997	0.9964	0.9819	0.9434	0.8689	0.7548	0.6098	0.4522	0.3036	0.1818	0.0951	0.0422	0.0152	0.0042	0.0008	0.0001	0.0000	0.0000
15	7	1.0000	1.0000	0.9994	0.9958	0.9827	0.9500	0.8868	0.7869	0.6535	0.5000	0.3465	0.2131	0.1132	0.0500	0.0173	0.0042	0.0006	0.0000	0.0000
15	8	1.0000	1.0000	0.9999	0.9992	0.9958	0.9848	0.9578	0.9050	0.8182	0.6964	0.5478	0.3902	0.2452	0.1311	0.0566	0.0181	0.0036	0.0003	0.0000
15	9	1.0000	1.0000	1.0000	0.9999	0.9992	0.9964	0.9876	0.9662	0.9231	0.8491	0.7392	0.5968	0.4357	0.2784	0.1484	0.0611	0.0168	0.0023	0.0001
15	10	1.0000	1.0000	1.0000	1.0000	0.9999	0.9993	0.9972	0.9907	0.9745	0.9408	0.8796	0.7827	0.6481	0.4845	0.3135	0.1642	0.0617	0.0127	0.0006
15	11	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9995	0.9981	0.9937	0.9824	0.9576	0.9095	0.8273	0.7031	0.5387	0.3518	0.1773	0.0556	0.0055
15	12	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9997	0.9989	0.9963	0.9894	0.9729	0.9383	0.8732	0.7639	0.6020	0.3958	0.1841	0.0362
15	13	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9995	0.9983	0.9948	0.9858	0.9647	0.9198	0.8329	0.6814	0.4510	0.1710
15	14	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9995	0.9984	0.9953	0.9866	0.9648	0.9127	0.7941	0.5367
16	0	0.4401	0.1853	0.0743	0.0282	0.0100	0.0033	0.0010	0.0003	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
16	1	0.8108	0.5147	0.2839	0.1407	0.0635	0.0261	0.0098	0.0033	0.0010	0.0003	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
16	2	0.9571	0.7893	0.5614	0.3518	0.1971	0.0994	0.0451	0.0183	0.0066	0.0021	0.0006	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
16	3	0.9930	0.9316	0.7899	0.5981	0.4050	0.2459	0.1339	0.0652	0.0281	0.0106	0.0035	0.0009	0.0002	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
16	4	0.9991	0.9830	0.9210	0.7983	0.6302	0.4499	0.2892	0.1666	0.0853	0.0384	0.0149	0.0049	0.0013	0.0003	0.0000	0.0000	0.0000	0.0000	0.0000
16	5	0.9999	0.9967	0.9765	0.9183	0.8104	0.6598	0.4900	0.3288	0.1976	0.1051	0.0486	0.0191	0.0062	0.0016	0.0003	0.0000	0.0000	0.0000	0.0000
16	6	1.0000	0.9995	0.9944	0.9733	0.9204	0.8247	0.6882	0.5272	0.3660	0.2273	0.1241	0.0583	0.0229	0.0071	0.0016	0.0003	0.0000	0.0000	0.0000
16	7	1.0000	0.9999	0.9989	0.9930	0.9729	0.9257	0.8406	0.7161	0.5629	0.4018	0.2559	0.1423	0.0671	0.0257	0.0075	0.0015	0.0002	0.0000	0.0000
16	8	1.0000	1.0000	0.9998	0.9985	0.9925	0.9743	0.9329	0.8577	0.7441	0.5982	0.4371	0.2839	0.1594	0.0744	0.0271	0.0070	0.0011	0.0001	0.0000
16	9	1.0000	1.0000	1.0000	0.9998	0.9984	0.9929	0.9771	0.9417	0.8759	0.7728	0.6340	0.4728	0.3119	0.1753	0.0796	0.0267	0.0056	0.0005	0.0000
16	10	1.0000	1.0000	1.0000	1.0000	0.9997	0.9984	0.9938	0.9809	0.9514	0.8949	0.8024	0.6712	0.5100	0.3402	0.1897	0.0817	0.0235	0.0033	0.0001
16	11	1.0000	1.0000	1.0000	1.0000	1.0000	0.9997	0.9987	0.9951	0.9851	0.9616	0.9147	0.8334	0.7108	0.5501	0.3698	0.2018	0.0791	0.0170	0.0009
16	12	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9998	0.9991	0.9965	0.9894	0.9719	0.9349	0.8661	0.7541	0.5950	0.4019	0.2101	0.0684	0.0070
16	13	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9994	0.9979	0.9934	0.9817	0.9549	0.9006	0.8029	0.6482	0.4386	0.2108	0.0429
16	14	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9997	0.9990	0.9967	0.9902	0.9739	0.9365	0.8593	0.7161	0.4853	0.1892
16	15	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9997	0.9990	0.9967	0.9900	0.9719	0.9258	0.8147	0.5599
17	0	0.4181	0.1668	0.0631	0.0225	0.0075	0.0023	0.0007	0.0002	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

（续表）

n	m	P																		
		0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90	0.95
17	1	0.7922	0.4818	0.2525	0.1182	0.0501	0.0193	0.0067	0.0021	0.0006	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
17	2	0.9498	0.7618	0.5198	0.3096	0.1637	0.0774	0.0327	0.0123	0.0041	0.0012	0.0003	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
17	3	0.9912	0.9174	0.7556	0.5489	0.3530	0.2019	0.1028	0.0464	0.0185	0.0064	0.0019	0.0005	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
17	4	0.9988	0.9779	0.9013	0.7582	0.5739	0.3887	0.2348	0.1260	0.0596	0.0245	0.0086	0.0025	0.0006	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000
17	5	0.9999	0.9953	0.9681	0.8943	0.7653	0.5968	0.4197	0.2639	0.1471	0.0717	0.0301	0.0106	0.0030	0.0007	0.0001	0.0000	0.0000	0.0000	0.0000
17	6	1.0000	0.9992	0.9917	0.9623	0.8929	0.7752	0.6188	0.4478	0.2902	0.1662	0.0826	0.0348	0.0120	0.0032	0.0006	0.0001	0.0000	0.0000	0.0000
17	7	1.0000	0.9999	0.9983	0.9891	0.9598	0.8954	0.7872	0.6405	0.4743	0.3145	0.1834	0.0919	0.0383	0.0127	0.0031	0.0005	0.0000	0.0000	0.0000
17	8	1.0000	1.0000	0.9997	0.9974	0.9876	0.9597	0.9006	0.8011	0.6626	0.5000	0.3374	0.1989	0.0994	0.0403	0.0124	0.0026	0.0003	0.0000	0.0000
17	9	1.0000	1.0000	1.0000	0.9995	0.9969	0.9873	0.9617	0.9081	0.8166	0.6855	0.5257	0.3595	0.2128	0.1046	0.0402	0.0109	0.0017	0.0001	0.0000
17	10	1.0000	1.0000	1.0000	0.9999	0.9994	0.9968	0.9880	0.9652	0.9174	0.8339	0.7098	0.5522	0.3812	0.2248	0.1071	0.0377	0.0083	0.0008	0.0000
17	11	1.0000	1.0000	1.0000	1.0000	0.9999	0.9993	0.9970	0.9894	0.9699	0.9283	0.8529	0.7361	0.5803	0.4032	0.2347	0.1057	0.0319	0.0047	0.0001
17	12	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9994	0.9975	0.9914	0.9755	0.9404	0.8740	0.7652	0.6113	0.4261	0.2418	0.0987	0.0221	0.0012
17	13	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9996	0.9981	0.9936	0.9816	0.9536	0.8972	0.7981	0.6470	0.4511	0.2444	0.0826	0.0088
17	14	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9997	0.9988	0.9959	0.9877	0.9673	0.9226	0.8363	0.6904	0.4802	0.2382	0.0503
17	15	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9994	0.9979	0.9933	0.9807	0.9499	0.8818	0.7476	0.5182	0.2078
17	16	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9998	0.9993	0.9977	0.9925	0.9775	0.9369	0.8332	0.5819
18	0	0.3972	0.1501	0.0537	0.0180	0.0056	0.0016	0.0004	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
18	1	0.7735	0.4503	0.2241	0.0991	0.0395	0.0142	0.0046	0.0013	0.0003	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
18	2	0.9419	0.7338	0.4797	0.2713	0.1353	0.0600	0.0236	0.0082	0.0025	0.0007	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
18	3	0.9891	0.9018	0.7202	0.5010	0.3057	0.1646	0.0783	0.0328	0.0120	0.0038	0.0010	0.0002	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
18	4	0.9985	0.9718	0.8794	0.7164	0.5187	0.3327	0.1886	0.0942	0.0411	0.0154	0.0049	0.0013	0.0003	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
18	5	0.9998	0.9936	0.9581	0.8671	0.7175	0.5344	0.3550	0.2088	0.1077	0.0481	0.0183	0.0058	0.0014	0.0003	0.0000	0.0000	0.0000	0.0000	0.0000
18	6	1.0000	0.9988	0.9882	0.9487	0.8610	0.7217	0.5491	0.3743	0.2258	0.1189	0.0537	0.0203	0.0062	0.0014	0.0002	0.0000	0.0000	0.0000	0.0000
18	7	1.0000	0.9998	0.9973	0.9837	0.9431	0.8593	0.7283	0.5634	0.3915	0.2403	0.1280	0.0577	0.0212	0.0061	0.0012	0.0002	0.0000	0.0000	0.0000
18	8	1.0000	1.0000	0.9995	0.9958	0.9807	0.9404	0.8609	0.7368	0.5779	0.4073	0.2527	0.1347	0.0597	0.0210	0.0054	0.0009	0.0001	0.0000	0.0000
18	9	1.0000	1.0000	0.9999	0.9991	0.9946	0.9790	0.9403	0.8653	0.7473	0.5927	0.4222	0.2632	0.1391	0.0596	0.0194	0.0043	0.0005	0.0000	0.0000

（续表）

n	m	P 0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90	0.95
18	10	1.0000	1.0000	1.0000	0.9998	0.9988	0.9939	0.9788	0.9424	0.8720	0.7597	0.6085	0.4366	0.2717	0.1407	0.0570	0.0163	0.0027	0.0002	0.0000
18	11	1.0000	1.0000	1.0000	1.0000	0.9998	0.9986	0.9938	0.9797	0.9463	0.8811	0.7742	0.6257	0.4509	0.2783	0.1390	0.0513	0.0118	0.0012	0.0000
18	12	1.0000	1.0000	1.0000	1.0000	1.0000	0.9997	0.9986	0.9943	0.9817	0.9519	0.8923	0.7912	0.6450	0.4656	0.2826	0.1329	0.0419	0.0064	0.0002
18	13	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9997	0.9987	0.9951	0.9846	0.9589	0.9058	0.8114	0.6674	0.4813	0.2837	0.1206	0.0282	0.0016
18	14	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9998	0.9990	0.9962	0.9880	0.9672	0.9217	0.8355	0.6943	0.4990	0.2798	0.0982	0.0109
18	15	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9993	0.9975	0.9918	0.9764	0.9401	0.8647	0.7287	0.5203	0.2662	0.0581
18	16	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9997	0.9987	0.9954	0.9858	0.9605	0.9009	0.7760	0.5497	0.2265
18	17	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9996	0.9984	0.9944	0.9820	0.9464	0.8499	0.6028
19	0	0.3774	0.1351	0.0456	0.0144	0.0042	0.0011	0.0003	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
19	1	0.7547	0.4203	0.1985	0.0829	0.0310	0.0104	0.0031	0.0008	0.0002	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
19	2	0.9335	0.7054	0.4413	0.2369	0.1113	0.0462	0.0170	0.0055	0.0015	0.0004	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
19	3	0.9868	0.8850	0.6842	0.4551	0.2631	0.1332	0.0591	0.0230	0.0077	0.0022	0.0005	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
19	4	0.9980	0.9648	0.8556	0.6733	0.4654	0.2822	0.1500	0.0696	0.0280	0.0096	0.0028	0.0006	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
19	5	0.9998	0.9914	0.9463	0.8369	0.6678	0.4739	0.2968	0.1629	0.0777	0.0318	0.0109	0.0031	0.0007	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000
19	6	1.0000	0.9983	0.9837	0.9324	0.8251	0.6655	0.4812	0.3081	0.1727	0.0835	0.0342	0.0116	0.0031	0.0006	0.0001	0.0000	0.0000	0.0000	0.0000
19	7	1.0000	0.9997	0.9959	0.9767	0.9225	0.8180	0.6656	0.4878	0.3169	0.1796	0.0871	0.0352	0.0114	0.0028	0.0005	0.0001	0.0000	0.0000	0.0000
19	8	1.0000	1.0000	0.9992	0.9933	0.9713	0.9161	0.8145	0.6675	0.4940	0.3238	0.1841	0.0885	0.0347	0.0105	0.0023	0.0003	0.0000	0.0000	0.0000
19	9	1.0000	1.0000	0.9999	0.9984	0.9911	0.9675	0.9125	0.8139	0.6710	0.5000	0.3290	0.1861	0.0875	0.0326	0.0089	0.0016	0.0001	0.0000	0.0000
19	10	1.0000	1.0000	1.0000	0.9997	0.9977	0.9895	0.9653	0.9115	0.8159	0.6762	0.5060	0.3325	0.1855	0.0839	0.0288	0.0067	0.0008	0.0000	0.0000
19	11	1.0000	1.0000	1.0000	1.0000	0.9995	0.9972	0.9886	0.9648	0.9129	0.8204	0.6831	0.5122	0.3344	0.1820	0.0775	0.0233	0.0041	0.0003	0.0000
19	12	1.0000	1.0000	1.0000	1.0000	0.9999	0.9994	0.9969	0.9884	0.9658	0.9165	0.8273	0.6919	0.5188	0.3345	0.1749	0.0676	0.0163	0.0017	0.0000
19	13	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9993	0.9969	0.9891	0.9682	0.9223	0.8371	0.7032	0.5261	0.3322	0.1631	0.0537	0.0086	0.0002
19	14	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9994	0.9972	0.9904	0.9720	0.9304	0.8500	0.7178	0.5346	0.3267	0.1444	0.0352	0.0020
19	15	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9995	0.9978	0.9923	0.9770	0.9409	0.8668	0.7369	0.5449	0.3159	0.1150	0.0132
19	16	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9996	0.9985	0.9945	0.9830	0.9538	0.8887	0.7631	0.5587	0.2946	0.0666
19	17	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9998	0.9992	0.9969	0.9896	0.9690	0.9171	0.8015	0.5797	0.2453
19	18	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9997	0.9989	0.9958	0.9856	0.9544	0.8649	0.6227
20	0	0.3585	0.1216	0.0388	0.0115	0.0032	0.0008	0.0002	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

（续表）

n	m	P																		
		0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90	0.95
20	1	0.7358	0.3918	0.1756	0.0692	0.0243	0.0076	0.0021	0.0005	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
20	2	0.9245	0.6769	0.4049	0.2061	0.0913	0.0355	0.0121	0.0036	0.0009	0.0002	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
20	3	0.9841	0.8671	0.6477	0.4115	0.2252	0.1071	0.0444	0.0160	0.0049	0.0013	0.0003	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
20	4	0.9974	0.9568	0.8299	0.6297	0.4148	0.2375	0.1182	0.0510	0.0189	0.0059	0.0015	0.0003	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
20	5	0.9997	0.9888	0.9327	0.8042	0.6172	0.4164	0.2454	0.1256	0.0553	0.0207	0.0064	0.0016	0.0003	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
20	6	1.0000	0.9976	0.9781	0.9133	0.7858	0.6080	0.4166	0.2500	0.1299	0.0577	0.0214	0.0065	0.0015	0.0003	0.0000	0.0000	0.0000	0.0000	0.0000
20	7	1.0000	0.9996	0.9941	0.9679	0.8982	0.7723	0.6010	0.4159	0.2520	0.1316	0.0580	0.0210	0.0060	0.0013	0.0002	0.0000	0.0000	0.0000	0.0000
20	8	1.0000	0.9999	0.9987	0.9900	0.9591	0.8867	0.7624	0.5956	0.4143	0.2517	0.1308	0.0565	0.0196	0.0051	0.0009	0.0001	0.0000	0.0000	0.0000
20	9	1.0000	1.0000	0.9998	0.9974	0.9861	0.9520	0.8782	0.7553	0.5914	0.4119	0.2493	0.1275	0.0532	0.0171	0.0039	0.0006	0.0000	0.0000	0.0000
20	10	1.0000	1.0000	1.0000	0.9994	0.9961	0.9829	0.9468	0.8725	0.7507	0.5881	0.4086	0.2447	0.1218	0.0480	0.0139	0.0026	0.0003	0.0000	0.0000
20	11	1.0000	1.0000	1.0000	0.9999	0.9991	0.9949	0.9804	0.9435	0.8692	0.7483	0.5857	0.4044	0.2376	0.1133	0.0409	0.0100	0.0013	0.0001	0.0000
20	12	1.0000	1.0000	1.0000	1.0000	0.9998	0.9987	0.9940	0.9790	0.9420	0.8684	0.7480	0.5841	0.3990	0.2277	0.1018	0.0321	0.0059	0.0004	0.0000
20	13	1.0000	1.0000	1.0000	1.0000	1.0000	0.9997	0.9985	0.9935	0.9786	0.9423	0.8701	0.7500	0.5834	0.3920	0.2142	0.0867	0.0219	0.0024	0.0000
20	14	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9997	0.9984	0.9936	0.9793	0.9447	0.8744	0.7546	0.5836	0.3828	0.1958	0.0673	0.0113	0.0003
20	15	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9997	0.9985	0.9941	0.9811	0.9491	0.8818	0.7625	0.5852	0.3704	0.1702	0.0432	0.0026
20	16	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9997	0.9987	0.9951	0.9840	0.9556	0.8929	0.7748	0.5886	0.3523	0.1330	0.0159
20	17	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9998	0.9991	0.9964	0.9879	0.9645	0.9087	0.7939	0.5951	0.3231	0.0755
20	18	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9995	0.9979	0.9924	0.9757	0.9308	0.8244	0.6083	0.2642
20	19	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9998	0.9992	0.9968	0.9885	0.9612	0.8784	0.6415

附 录 2

泊松分布累积概率表，查表参数为泊松事件发生率（期望值）λ 和发生次数 t

$$F_x(t)=P(x\leqslant t)=\sum_{x=0}^{t}\left(\frac{e^{-\lambda}\lambda^x}{x!}\right)$$

部分文献的泊松分布表是基于 $1-F_x(t-1)=\sum_{x=t}^{\infty}\left(\frac{e^{-\lambda}\lambda^x}{x!}\right)$ 制作，查表时注意公式说明。

程序28-2　生成泊松分布的累积概率表

```
data ProbPos;
  do t= 0 to 9;
    array c[20];
    i=1;
    do lambda=0.1 to 2 by 0.1;
      c[i]=cdf ('POISSON', t, lambda);
      i=i+1;
    end;
    format c: 9.4;
    keep t c: ;
    output;
  end;
run;
/* 创建列标签 1 - 2 并打印*/
%macro SetLabel();
  data ProbPos;
    set ProbPos;
    %let i=1;
    %do i=1 %to 20;
      %let p= %sysfunc(putn(%sysevalf( &i * 0.1), 4.1));
      label c&i="&p";
    %end;
  run;
%mend;
%SetLabel();
proc print label;run;
```

运行程序输出如下泊松分布累积概率表。第 1 行为均值参数，即期望值 λ，表示单位时间内的发生率；第 1 列为整数随机变量 t，表示多少个时间单位。根据它们查表所得单元格的值就是该发生率 λ 下，泊松事件成功发生次数小于等于 t 次的概率。

比如单位时间内出生的婴儿数 λ 为 3，则接下来的 1 个时间单位 $(t-1)$ 内出生小于等于 1 个婴儿（0 个婴儿或 1 个婴儿）的概率就是查表所得的概率值 0.1991。据此可推断接下来的 1 个时间单位内至少出生 2 个婴儿的概率就等于 1　0.1991　0.8009 即 80.09%。

表 28-2　泊松分布累积概率表

t	λ																			
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2.0
0	0.9048	0.8187	0.7408	0.6703	0.6065	0.5488	0.4966	0.4493	0.4066	0.3679	0.3329	0.3012	0.2725	0.2466	0.2231	0.2019	0.1827	0.1653	0.1496	0.1353
1	0.9953	0.9825	0.9631	0.9384	0.9098	0.8781	0.8442	0.8088	0.7725	0.7358	0.6990	0.6626	0.6268	0.5918	0.5578	0.5249	0.4932	0.4628	0.4337	0.4060
2	0.9998	0.9989	0.9964	0.9921	0.9856	0.9769	0.9659	0.9526	0.9371	0.9197	0.9004	0.8795	0.8571	0.8335	0.8088	0.7834	0.7572	0.7306	0.7037	0.6767
3	1.0000	0.9999	0.9997	0.9992	0.9982	0.9966	0.9942	0.9909	0.9865	0.9810	0.9743	0.9662	0.9569	0.9463	0.9344	0.9212	0.9068	0.8913	0.8747	0.8571
4	1.0000	1.0000	1.0000	0.9999	0.9998	0.9996	0.9992	0.9986	0.9977	0.9963	0.9946	0.9923	0.9893	0.9857	0.9814	0.9763	0.9704	0.9636	0.9559	0.9473
5	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9998	0.9997	0.9994	0.9990	0.9985	0.9978	0.9968	0.9955	0.9940	0.9920	0.9896	0.9868	0.9834
6	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9999	0.9997	0.9996	0.9994	0.9991	0.9987	0.9981	0.9974	0.9966	0.9955
7	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9999	0.9998	0.9997	0.9996	0.9994	0.9992	0.9989
8	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9999	0.9998	0.9998
9	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

t	2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9	3.0	3.1	3.2	3.3	3.4	3.5	3.6	3.7	3.8	3.9	4.0
0	0.1225	0.1108	0.1003	0.0907	0.0821	0.0743	0.0672	0.0608	0.0550	0.0498	0.0450	0.0408	0.0369	0.0334	0.0302	0.0273	0.0247	0.0224	0.0202	0.0183
1	0.3796	0.3546	0.3309	0.3084	0.2873	0.2674	0.2487	0.2311	0.2146	0.1991	0.1847	0.1712	0.1586	0.1468	0.1359	0.1257	0.1162	0.1074	0.0992	0.0916
2	0.6496	0.6227	0.5960	0.5697	0.5438	0.5184	0.4936	0.4695	0.4460	0.4232	0.4012	0.3799	0.3594	0.3397	0.3208	0.3027	0.2854	0.2689	0.2531	0.2381
3	0.8386	0.8194	0.7993	0.7787	0.7576	0.7360	0.7141	0.6919	0.6696	0.6472	0.6248	0.6025	0.5803	0.5584	0.5366	0.5152	0.4942	0.4735	0.4532	0.4335
4	0.9379	0.9275	0.9162	0.9041	0.8912	0.8774	0.8629	0.8477	0.8318	0.8153	0.7982	0.7806	0.7626	0.7442	0.7254	0.7064	0.6872	0.6678	0.6484	0.6288
5	0.9796	0.9751	0.9700	0.9643	0.9580	0.9510	0.9433	0.9349	0.9258	0.9161	0.9057	0.8946	0.8829	0.8705	0.8576	0.8441	0.8301	0.8156	0.8006	0.7851
6	0.9941	0.9925	0.9906	0.9884	0.9858	0.9828	0.9794	0.9756	0.9713	0.9665	0.9612	0.9554	0.9490	0.9421	0.9347	0.9267	0.9182	0.9091	0.8995	0.8893
7	0.9985	0.9980	0.9974	0.9967	0.9958	0.9947	0.9934	0.9919	0.9901	0.9881	0.9858	0.9832	0.9802	0.9769	0.9733	0.9692	0.9648	0.9599	0.9546	0.9489
8	0.9997	0.9995	0.9994	0.9991	0.9989	0.9985	0.9981	0.9976	0.9969	0.9962	0.9953	0.9943	0.9931	0.9917	0.9901	0.9883	0.9863	0.9840	0.9815	0.9786
9	0.9999	0.9999	0.9999	0.9998	0.9997	0.9996	0.9995	0.9993	0.9991	0.9989	0.9986	0.9982	0.9978	0.9973	0.9967	0.9960	0.9952	0.9942	0.9931	0.9919
10	1.0000	1.0000	1.0000	1.0000	0.9999	0.9999	0.9999	0.9998	0.9998	0.9997	0.9996	0.9995	0.9994	0.9992	0.9990	0.9987	0.9984	0.9981	0.9977	0.9972
11	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9999	0.9999	0.9999	0.9998	0.9998	0.9997	0.9996	0.9995	0.9994	0.9993	0.9991
12	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9999	0.9999	0.9999	0.9998	0.9998	0.9997
13	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9999
14	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

（续表）

t	4.1	4.2	4.3	4.4	4.5	4.6	4.7	4.8	4.9	5.0	5.1	5.2	5.3	5.4	5.5	5.6	5.7	5.8	5.9	6.0
0	0.0166	0.0150	0.0136	0.0123	0.0111	0.0101	0.0091	0.0082	0.0074	0.0067	0.0061	0.0055	0.0050	0.0045	0.0041	0.0037	0.0033	0.0030	0.0027	0.0025
1	0.0845	0.0780	0.0719	0.0663	0.0611	0.0563	0.0518	0.0477	0.0439	0.0404	0.0372	0.0342	0.0314	0.0289	0.0266	0.0244	0.0224	0.0206	0.0189	0.0174
2	0.2238	0.2102	0.1974	0.1851	0.1736	0.1626	0.1523	0.1425	0.1333	0.1247	0.1165	0.1088	0.1016	0.0948	0.0884	0.0824	0.0768	0.0715	0.0666	0.0620
3	0.4142	0.3954	0.3772	0.3594	0.3423	0.3257	0.3097	0.2942	0.2793	0.2650	0.2513	0.2381	0.2254	0.2133	0.2017	0.1906	0.1800	0.1700	0.1604	0.1512
4	0.6093	0.5898	0.5704	0.5512	0.5321	0.5132	0.4946	0.4763	0.4582	0.4405	0.4231	0.4061	0.3895	0.3733	0.3575	0.3422	0.3272	0.3127	0.2987	0.2851
5	0.7693	0.7531	0.7367	0.7199	0.7029	0.6858	0.6684	0.6510	0.6335	0.6160	0.5984	0.5809	0.5635	0.5461	0.5289	0.5119	0.4950	0.4783	0.4619	0.4457
6	0.8786	0.8675	0.8558	0.8436	0.8311	0.8180	0.8046	0.7908	0.7767	0.7622	0.7474	0.7324	0.7171	0.7017	0.6860	0.6703	0.6544	0.6384	0.6224	0.6063
7	0.9427	0.9361	0.9290	0.9214	0.9134	0.9049	0.8960	0.8867	0.8769	0.8666	0.8560	0.8449	0.8335	0.8217	0.8095	0.7970	0.7841	0.7710	0.7576	0.7440
8	0.9755	0.9721	0.9683	0.9642	0.9597	0.9549	0.9497	0.9442	0.9382	0.9319	0.9252	0.9181	0.9106	0.9027	0.8944	0.8857	0.8766	0.8672	0.8574	0.8472
9	0.9905	0.9889	0.9871	0.9851	0.9829	0.9805	0.9778	0.9749	0.9717	0.9682	0.9644	0.9603	0.9559	0.9512	0.9462	0.9409	0.9352	0.9292	0.9228	0.9161
10	0.9966	0.9959	0.9952	0.9943	0.9933	0.9922	0.9910	0.9896	0.9880	0.9863	0.9844	0.9823	0.9800	0.9775	0.9747	0.9718	0.9686	0.9651	0.9614	0.9574
11	0.9989	0.9986	0.9983	0.9980	0.9976	0.9971	0.9966	0.9960	0.9953	0.9945	0.9937	0.9927	0.9916	0.9904	0.9890	0.9875	0.9859	0.9841	0.9821	0.9799
12	0.9997	0.9996	0.9995	0.9993	0.9992	0.9990	0.9988	0.9986	0.9983	0.9980	0.9976	0.9972	0.9967	0.9962	0.9955	0.9949	0.9941	0.9932	0.9922	0.9912
13	0.9999	0.9999	0.9998	0.9998	0.9997	0.9997	0.9996	0.9995	0.9994	0.9993	0.9992	0.9990	0.9988	0.9986	0.9983	0.9980	0.9977	0.9973	0.9969	0.9964
14	1.0000	1.0000	1.0000	0.9999	0.9999	0.9999	0.9999	0.9999	0.9998	0.9998	0.9997	0.9997	0.9996	0.9995	0.9994	0.9993	0.9991	0.9990	0.9988	0.9986
15	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9999	0.9999	0.9999	0.9999	0.9998	0.9998	0.9998	0.9997	0.9996	0.9996	0.9995
16	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9998
17	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999
18	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

（续表）

t	6.1	6.2	6.3	6.4	6.5	6.6	6.7	6.8	6.9	7.0	7.1	7.2	7.3	7.4	7.5	7.6	7.7	7.8	7.9	8.0
0	0.0022	0.0020	0.0018	0.0017	0.0015	0.0014	0.0012	0.0011	0.0010	0.0009	0.0008	0.0007	0.0007	0.0006	0.0006	0.0005	0.0005	0.0004	0.0004	0.0003
1	0.0159	0.0146	0.0134	0.0123	0.0113	0.0103	0.0095	0.0087	0.0080	0.0073	0.0067	0.0061	0.0056	0.0051	0.0047	0.0043	0.0039	0.0036	0.0033	0.0030
2	0.0577	0.0536	0.0498	0.0463	0.0430	0.0400	0.0371	0.0344	0.0320	0.0296	0.0275	0.0255	0.0236	0.0219	0.0203	0.0188	0.0174	0.0161	0.0149	0.0138
3	0.1425	0.1342	0.1264	0.1189	0.1118	0.1052	0.0988	0.0928	0.0871	0.0818	0.0767	0.0719	0.0674	0.0632	0.0591	0.0554	0.0518	0.0485	0.0453	0.0424
4	0.2719	0.2592	0.2469	0.2351	0.2237	0.2127	0.2022	0.1920	0.1823	0.1730	0.1641	0.1555	0.1473	0.1395	0.1321	0.1249	0.1181	0.1117	0.1055	0.0996
5	0.4298	0.4141	0.3988	0.3837	0.3690	0.3547	0.3406	0.3270	0.3137	0.3007	0.2881	0.2759	0.2640	0.2526	0.2414	0.2307	0.2203	0.2103	0.2006	0.1912
6	0.5902	0.5742	0.5582	0.5423	0.5265	0.5108	0.4953	0.4799	0.4647	0.4497	0.4349	0.4204	0.4060	0.3920	0.3782	0.3646	0.3514	0.3384	0.3257	0.3134
7	0.7301	0.7160	0.7017	0.6873	0.6728	0.6581	0.6433	0.6285	0.6136	0.5987	0.5838	0.5689	0.5541	0.5393	0.5246	0.5100	0.4956	0.4812	0.4670	0.4530
8	0.8367	0.8259	0.8148	0.8033	0.7916	0.7796	0.7673	0.7548	0.7420	0.7291	0.7160	0.7027	0.6892	0.6757	0.6620	0.6482	0.6343	0.6204	0.6065	0.5925
9	0.9090	0.9016	0.8939	0.8858	0.8774	0.8686	0.8596	0.8502	0.8405	0.8305	0.8202	0.8096	0.7988	0.7877	0.7764	0.7649	0.7531	0.7411	0.7290	0.7166
10	0.9531	0.9486	0.9437	0.9386	0.9332	0.9274	0.9214	0.9151	0.9084	0.9015	0.8942	0.8867	0.8788	0.8707	0.8622	0.8535	0.8445	0.8352	0.8257	0.8159
11	0.9776	0.9750	0.9723	0.9693	0.9661	0.9627	0.9591	0.9552	0.9510	0.9467	0.9420	0.9371	0.9319	0.9265	0.9208	0.9148	0.9085	0.9020	0.8952	0.8881
12	0.9900	0.9887	0.9873	0.9857	0.9840	0.9821	0.9801	0.9779	0.9755	0.9730	0.9703	0.9673	0.9642	0.9609	0.9573	0.9536	0.9496	0.9454	0.9409	0.9362
13	0.9958	0.9952	0.9945	0.9937	0.9929	0.9920	0.9909	0.9898	0.9885	0.9872	0.9857	0.9841	0.9824	0.9805	0.9784	0.9762	0.9739	0.9714	0.9687	0.9658
14	0.9984	0.9981	0.9978	0.9974	0.9970	0.9966	0.9961	0.9956	0.9950	0.9943	0.9935	0.9927	0.9918	0.9908	0.9897	0.9886	0.9873	0.9859	0.9844	0.9827
15	0.9994	0.9993	0.9992	0.9990	0.9988	0.9986	0.9984	0.9982	0.9979	0.9976	0.9972	0.9969	0.9964	0.9959	0.9954	0.9948	0.9941	0.9934	0.9926	0.9918
16	0.9998	0.9997	0.9997	0.9996	0.9996	0.9995	0.9994	0.9993	0.9992	0.9990	0.9989	0.9987	0.9985	0.9983	0.9980	0.9978	0.9974	0.9971	0.9967	0.9963
17	0.9999	0.9999	0.9999	0.9999	0.9998	0.9998	0.9998	0.9997	0.9997	0.9996	0.9996	0.9995	0.9994	0.9993	0.9992	0.9991	0.9989	0.9988	0.9986	0.9984
18	1.0000	1.0000	1.0000	1.0000	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9998	0.9998	0.9998	0.9997	0.9997	0.9996	0.9996	0.9995	0.9994	0.9993
19	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9998	0.9998	0.9998	0.9997
20	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9999	0.9999	0.9999
21	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

（续表）

t	8.1	8.2	8.3	8.4	8.5	8.6	8.7	8.8	8.9	9.0	9.1	9.2	9.3	9.4	9.5	9.6	9.7	9.8	9.9	10.0
0	0.0003	0.0003	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0000
1	0.0028	0.0025	0.0023	0.0021	0.0019	0.0018	0.0016	0.0015	0.0014	0.0012	0.0011	0.0010	0.0009	0.0009	0.0008	0.0007	0.0007	0.0006	0.0005	0.0005
2	0.0127	0.0118	0.0109	0.0100	0.0093	0.0086	0.0079	0.0073	0.0068	0.0062	0.0058	0.0053	0.0049	0.0045	0.0042	0.0038	0.0035	0.0033	0.0030	0.0028
3	0.0396	0.0370	0.0346	0.0323	0.0301	0.0281	0.0262	0.0244	0.0228	0.0212	0.0198	0.0184	0.0172	0.0160	0.0149	0.0138	0.0129	0.0120	0.0111	0.0103
4	0.0940	0.0887	0.0837	0.0789	0.0744	0.0701	0.0660	0.0621	0.0584	0.0550	0.0517	0.0486	0.0456	0.0429	0.0403	0.0378	0.0355	0.0333	0.0312	0.0293
5	0.1822	0.1736	0.1653	0.1573	0.1496	0.1422	0.1352	0.1284	0.1219	0.1157	0.1098	0.1041	0.0986	0.0935	0.0885	0.0838	0.0793	0.0750	0.0710	0.0671
6	0.3013	0.2896	0.2781	0.2670	0.2562	0.2457	0.2355	0.2256	0.2160	0.2068	0.1978	0.1892	0.1808	0.1727	0.1649	0.1574	0.1502	0.1433	0.1366	0.1301
7	0.4391	0.4254	0.4119	0.3987	0.3856	0.3728	0.3602	0.3478	0.3357	0.3239	0.3123	0.3010	0.2900	0.2792	0.2687	0.2584	0.2485	0.2388	0.2294	0.2202
8	0.5786	0.5647	0.5507	0.5369	0.5231	0.5094	0.4958	0.4823	0.4689	0.4557	0.4426	0.4296	0.4168	0.4042	0.3918	0.3796	0.3676	0.3558	0.3442	0.3328
9	0.7041	0.6915	0.6788	0.6659	0.6530	0.6400	0.6269	0.6137	0.6006	0.5874	0.5742	0.5611	0.5479	0.5349	0.5218	0.5089	0.4960	0.4832	0.4705	0.4579
10	0.8058	0.7955	0.7850	0.7743	0.7634	0.7522	0.7409	0.7294	0.7178	0.7060	0.6941	0.6820	0.6699	0.6576	0.6453	0.6329	0.6205	0.6080	0.5955	0.5830
11	0.8807	0.8731	0.8652	0.8571	0.8487	0.8400	0.8311	0.8220	0.8126	0.8030	0.7932	0.7832	0.7730	0.7626	0.7520	0.7412	0.7303	0.7193	0.7081	0.6968
12	0.9313	0.9261	0.9207	0.9150	0.9091	0.9029	0.8965	0.8898	0.8829	0.8758	0.8684	0.8607	0.8529	0.8448	0.8364	0.8279	0.8191	0.8101	0.8009	0.7916
13	0.9628	0.9595	0.9561	0.9524	0.9486	0.9445	0.9403	0.9358	0.9311	0.9261	0.9210	0.9156	0.9100	0.9042	0.8981	0.8919	0.8853	0.8786	0.8716	0.8645
14	0.9810	0.9791	0.9771	0.9749	0.9726	0.9701	0.9675	0.9647	0.9617	0.9585	0.9552	0.9517	0.9480	0.9441	0.9400	0.9357	0.9312	0.9265	0.9216	0.9165
15	0.9908	0.9898	0.9887	0.9875	0.9862	0.9848	0.9832	0.9816	0.9798	0.9780	0.9760	0.9738	0.9715	0.9691	0.9665	0.9638	0.9609	0.9579	0.9546	0.9513
16	0.9958	0.9953	0.9947	0.9941	0.9934	0.9926	0.9918	0.9909	0.9899	0.9889	0.9878	0.9865	0.9852	0.9838	0.9823	0.9806	0.9789	0.9770	0.9751	0.9730
17	0.9982	0.9979	0.9977	0.9973	0.9970	0.9966	0.9962	0.9957	0.9952	0.9947	0.9941	0.9934	0.9927	0.9919	0.9911	0.9902	0.9892	0.9881	0.9870	0.9857
18	0.9992	0.9991	0.9990	0.9989	0.9987	0.9985	0.9983	0.9981	0.9978	0.9976	0.9973	0.9969	0.9966	0.9962	0.9957	0.9952	0.9947	0.9941	0.9935	0.9928
19	0.9997	0.9997	0.9996	0.9995	0.9995	0.9994	0.9993	0.9992	0.9991	0.9989	0.9988	0.9986	0.9985	0.9983	0.9980	0.9978	0.9975	0.9972	0.9969	0.9965
20	0.9999	0.9999	0.9998	0.9998	0.9998	0.9998	0.9997	0.9997	0.9996	0.9996	0.9995	0.9994	0.9993	0.9992	0.9991	0.9990	0.9989	0.9987	0.9986	0.9984
21	1.0000	1.0000	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9998	0.9998	0.9998	0.9998	0.9997	0.9997	0.9996	0.9996	0.9995	0.9995	0.9994	0.9993
22	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9998	0.9998	0.9998	0.9997	0.9997
23	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999
24	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

（续表）

t	11.0	12.0	13.0	14.0	15.0	16.0	17.0	18.0	19.0	20.0	21.0	22.0	23.0	24.0	25.0	26.0	27.0	28.0	29.0	30.0
0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
1	0.0002	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2	0.0012	0.0005	0.0002	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
3	0.0049	0.0023	0.0011	0.0005	0.0002	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
4	0.0151	0.0076	0.0037	0.0018	0.0009	0.0004	0.0002	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
5	0.0375	0.0203	0.0107	0.0055	0.0028	0.0014	0.0007	0.0003	0.0002	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
6	0.0786	0.0458	0.0259	0.0142	0.0076	0.0040	0.0021	0.0010	0.0005	0.0003	0.0001	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
7	0.1432	0.0895	0.0540	0.0316	0.0180	0.0100	0.0054	0.0029	0.0015	0.0008	0.0004	0.0002	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
8	0.2320	0.1550	0.0998	0.0621	0.0374	0.0220	0.0126	0.0071	0.0039	0.0021	0.0011	0.0006	0.0003	0.0002	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000
9	0.3405	0.2424	0.1658	0.1094	0.0699	0.0433	0.0261	0.0154	0.0089	0.0050	0.0028	0.0015	0.0008	0.0004	0.0002	0.0001	0.0001	0.0000	0.0000	0.0000
10	0.4599	0.3472	0.2517	0.1757	0.1185	0.0774	0.0491	0.0304	0.0183	0.0108	0.0063	0.0035	0.0020	0.0011	0.0006	0.0003	0.0002	0.0001	0.0000	0.0000
11	0.5793	0.4616	0.3532	0.2600	0.1848	0.1270	0.0847	0.0549	0.0347	0.0214	0.0129	0.0076	0.0044	0.0025	0.0014	0.0008	0.0004	0.0002	0.0001	0.0001
12	0.6887	0.5760	0.4631	0.3585	0.2676	0.1931	0.1350	0.0917	0.0606	0.0390	0.0245	0.0151	0.0091	0.0054	0.0031	0.0018	0.0010	0.0006	0.0003	0.0002
13	0.7813	0.6815	0.5730	0.4644	0.3632	0.2745	0.2009	0.1426	0.0984	0.0661	0.0434	0.0278	0.0174	0.0107	0.0065	0.0038	0.0022	0.0013	0.0007	0.0004
14	0.8540	0.7720	0.6751	0.5704	0.4657	0.3675	0.2808	0.2081	0.1497	0.1049	0.0716	0.0477	0.0311	0.0198	0.0124	0.0076	0.0046	0.0027	0.0016	0.0009
15	0.9074	0.8444	0.7636	0.6694	0.5681	0.4667	0.3715	0.2867	0.2148	0.1565	0.1111	0.0769	0.0520	0.0344	0.0223	0.0142	0.0088	0.0054	0.0033	0.0019
16	0.9441	0.8987	0.8355	0.7559	0.6641	0.5660	0.4677	0.3751	0.2920	0.2211	0.1629	0.1170	0.0821	0.0563	0.0377	0.0248	0.0160	0.0101	0.0063	0.0039
17	0.9678	0.9370	0.8905	0.8272	0.7489	0.6593	0.5640	0.4686	0.3784	0.2970	0.2270	0.1690	0.1228	0.0871	0.0605	0.0411	0.0274	0.0179	0.0115	0.0073
18	0.9823	0.9626	0.9302	0.8826	0.8195	0.7423	0.6550	0.5622	0.4695	0.3814	0.3017	0.2325	0.1748	0.1283	0.0920	0.0646	0.0445	0.0300	0.0199	0.0129
19	0.9907	0.9787	0.9573	0.9235	0.8752	0.8122	0.7363	0.6509	0.5606	0.4703	0.3843	0.3060	0.2377	0.1803	0.1336	0.0968	0.0687	0.0478	0.0326	0.0219
20	0.9953	0.9884	0.9750	0.9521	0.9170	0.8682	0.8055	0.7307	0.6472	0.5591	0.4710	0.3869	0.3101	0.2426	0.1855	0.1387	0.1015	0.0727	0.0511	0.0353
21	0.9977	0.9939	0.9859	0.9712	0.9469	0.9108	0.8615	0.7991	0.7255	0.6437	0.5577	0.4716	0.3894	0.3139	0.2473	0.1905	0.1436	0.1060	0.0767	0.0544
22	0.9990	0.9970	0.9924	0.9833	0.9673	0.9418	0.9047	0.8551	0.7931	0.7206	0.6405	0.5564	0.4723	0.3917	0.3175	0.2517	0.1952	0.1483	0.1104	0.0806

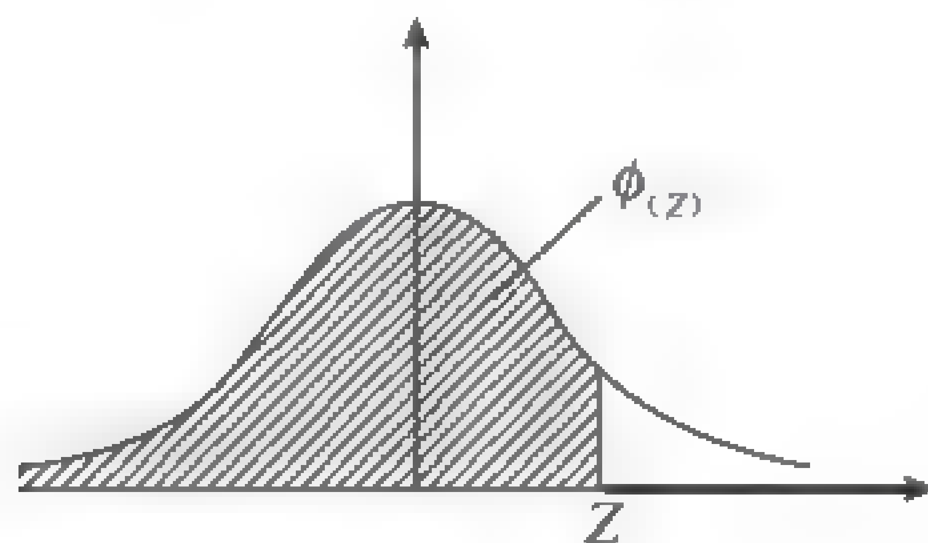
（续表）

t	11.0	12.0	13.0	14.0	15.0	16.0	17.0	18.0	19.0	20.0	21.0	22.0	23.0	24.0	25.0	26.0	27.0	28.0	29.0	30.0
23	0.9995	0.9985	0.9960	0.9907	0.9805	0.9633	0.9367	0.8989	0.8490	0.7875	0.7160	0.6374	0.5551	0.4728	0.3939	0.3209	0.2559	0.1998	0.1529	0.1146
24	0.9998	0.9993	0.9980	0.9950	0.9888	0.9777	0.9594	0.9317	0.8933	0.8432	0.7822	0.7117	0.6346	0.5540	0.4734	0.3959	0.3242	0.2599	0.2042	0.1572
25	0.9999	0.9997	0.9990	0.9974	0.9938	0.9869	0.9748	0.9554	0.9269	0.8878	0.8377	0.7771	0.7077	0.6319	0.5529	0.4739	0.3979	0.3272	0.2637	0.2084
26	1.0000	0.9999	0.9995	0.9987	0.9967	0.9925	0.9848	0.9718	0.9514	0.9221	0.8826	0.8324	0.7723	0.7038	0.6294	0.5519	0.4744	0.3997	0.3301	0.2673
27	1.0000	0.9999	0.9998	0.9994	0.9983	0.9959	0.9912	0.9827	0.9687	0.9475	0.9175	0.8775	0.8274	0.7677	0.7002	0.6270	0.5509	0.4749	0.4014	0.3329
28	1.0000	1.0000	0.9999	0.9997	0.9991	0.9978	0.9950	0.9897	0.9805	0.9657	0.9436	0.9129	0.8726	0.8225	0.7634	0.6967	0.6247	0.5500	0.4753	0.4031
29	1.0000	1.0000	1.0000	0.9999	0.9996	0.9989	0.9973	0.9941	0.9882	0.9782	0.9626	0.9398	0.9085	0.8679	0.8179	0.7593	0.6935	0.6226	0.5492	0.4757
30	1.0000	1.0000	1.0000	0.9999	0.9998	0.9994	0.9986	0.9967	0.9930	0.9865	0.9758	0.9595	0.9360	0.9042	0.8633	0.8134	0.7553	0.6903	0.6206	0.5484
31	1.0000	1.0000	1.0000	1.0000	0.9999	0.9997	0.9993	0.9982	0.9960	0.9919	0.9848	0.9735	0.9564	0.9322	0.8999	0.8589	0.8092	0.7515	0.6874	0.6186
32	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9996	0.9990	0.9978	0.9953	0.9907	0.9831	0.9711	0.9533	0.9285	0.8958	0.8546	0.8051	0.7479	0.6845
33	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9998	0.9995	0.9988	0.9973	0.9945	0.9895	0.9813	0.9686	0.9502	0.9249	0.8918	0.8505	0.8011	0.7444
34	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9998	0.9994	0.9985	0.9968	0.9936	0.9882	0.9794	0.9662	0.9472	0.9213	0.8879	0.8465	0.7973
35	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9997	0.9992	0.9982	0.9962	0.9927	0.9868	0.9775	0.9637	0.9441	0.9178	0.8841	0.8426
36	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9998	0.9996	0.9990	0.9978	0.9956	0.9918	0.9854	0.9756	0.9612	0.9411	0.9144	0.8804
37	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9998	0.9995	0.9988	0.9974	0.9950	0.9908	0.9840	0.9737	0.9587	0.9381	0.9110
38	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9997	0.9993	0.9985	0.9970	0.9943	0.9897	0.9825	0.9717	0.9562	0.9352
39	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9999	0.9996	0.9992	0.9983	0.9966	0.9936	0.9887	0.9810	0.9697	0.9537

附 录 3

标准正态分布累积概率表，查表参数为指定随机变量 z

$$\Phi(z)=P(Z\leqslant z)=\int_{-\infty}^{z}\frac{1}{\sqrt{2\pi}}\times e^{-\frac{z^2}{2}}dz$$



程序28-3 生成正态分布累积概率表

```
data ProbNorm;
  do z=-3.50 to 3.50 by 0.1;
    array c[10];
    i=1;
    do j=0.00 to 0.09 by 0.01;
      c[i]=ProbNorm(z+j); /*等价于 cdf( "NORMAL", z+j) */
      i=i+1;
    end;
    format c: 9.4;
    keep z c:;
    output;
  end;
run;
/*创建列标签 0.00 - 0.09 并打印*/
%macro SetLabel();
  data ProbNorm;
    set ProbNorm;
    %let i=1;
    %do i=1 %to 10;
      %let p= %sysfunc(putn(%sysevalf( (&i-1) * 0.01), 4.2));
      label c&i="&p";
    %end;
  run;
%mend;
%SetLabel();
proc print label;run;
```

运行程序输出如下标准正态分布累积概率表。第1列为随机变量 z 的取值 -3.5～3.5，第1行是小数点后第2位的取值（0.01～0.09），因此每一个单元格是水平方向上的 z 值加上对应列的小数 j 所指定的分位数的累积概率值。也就是说该表从左到右，从上到下服从标准正态分布的随机变量每增长0.01所对应的累积概率值。对于所有正态分布需要化为标准正态分布才可以查表。

比如：已知随机变量服从均值为 5，方差为 9 的正态分布，记为 $X \sim N(5, 9)$，求该随机变量大于等于 11 且小于等于 -1 的概率，即 $P(X \geqslant 11)$ 且 $P(X \leqslant -1)$ 的概率？

解答：对任意 $X \sim N(\mu, \sigma^2)$ 可据 $z = (x - \mu)/\sigma \sim N(0,1)$ 化为标准正态分布。则 $z = (11 - 5)/3 = 2$，$z = (-1 - 5)/3 = -2$，则它等价于求标准正态分布中 $P(|Z| \geqslant 2)$ 的概率。

由于 $P(|Z| \geqslant 2) = 2 \times P(Z \geqslant 2)$，而 $P(Z \geqslant 2) = 1 - P(Z \leqslant 2)$，而 $P(Z \leqslant 2)$ 可直接查标准正态分布累积概率表第 56 行（2.0 行）的第 2 列（0.00 列），其取值是 0.9772，即 $P(Z \leqslant 2.00) = 0.9772$，则 $P(|Z| \geqslant 2) = 2 \times P(Z \geqslant 2) = 2 \times (1 - 0.9772) = 0.0456$。

表 28-3　标准正态分布累积概率表

z	j									
	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
-3.5	0.0002	0.0002	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003
-3.4	0.0003	0.0003	0.0004	0.0004	0.0004	0.0004	0.0004	0.0004	0.0005	0.0005
-3.3	0.0005	0.0005	0.0005	0.0005	0.0006	0.0006	0.0006	0.0006	0.0006	0.0007
-3.2	0.0007	0.0007	0.0007	0.0008	0.0008	0.0008	0.0008	0.0009	0.0009	0.0009
-3.1	0.0010	0.0010	0.0010	0.0011	0.0011	0.0011	0.0012	0.0012	0.0013	0.0013
-3.0	0.0013	0.0014	0.0014	0.0015	0.0015	0.0016	0.0016	0.0017	0.0018	0.0018
-2.9	0.0019	0.0019	0.0020	0.0021	0.0021	0.0022	0.0023	0.0023	0.0024	0.0025
-2.8	0.0026	0.0026	0.0027	0.0028	0.0029	0.0030	0.0031	0.0032	0.0033	0.0034
-2.7	0.0035	0.0036	0.0037	0.0038	0.0039	0.0040	0.0041	0.0043	0.0044	0.0045
-2.6	0.0047	0.0048	0.0049	0.0051	0.0052	0.0054	0.0055	0.0057	0.0059	0.0060
-2.5	0.0062	0.0064	0.0066	0.0068	0.0069	0.0071	0.0073	0.0075	0.0078	0.0080
-2.4	0.0082	0.0084	0.0087	0.0089	0.0091	0.0094	0.0096	0.0099	0.0102	0.0104
-2.3	0.0107	0.0110	0.0113	0.0116	0.0119	0.0122	0.0125	0.0129	0.0132	0.0136
-2.2	0.0139	0.0143	0.0146	0.0150	0.0154	0.0158	0.0162	0.0166	0.0170	0.0174
-2.1	0.0179	0.0183	0.0188	0.0192	0.0197	0.0202	0.0207	0.0212	0.0217	0.0222
-2.0	0.0228	0.0233	0.0239	0.0244	0.0250	0.0256	0.0262	0.0268	0.0274	0.0281
-1.9	0.0287	0.0294	0.0301	0.0307	0.0314	0.0322	0.0329	0.0336	0.0344	0.0351
-1.8	0.0359	0.0367	0.0375	0.0384	0.0392	0.0401	0.0409	0.0418	0.0427	0.0436
-1.7	0.0446	0.0455	0.0465	0.0475	0.0485	0.0495	0.0505	0.0516	0.0526	0.0537
-1.6	0.0548	0.0559	0.0571	0.0582	0.0594	0.0606	0.0618	0.0630	0.0643	0.0655
-1.5	0.0668	0.0681	0.0694	0.0708	0.0721	0.0735	0.0749	0.0764	0.0778	0.0793
-1.4	0.0808	0.0823	0.0838	0.0853	0.0869	0.0885	0.0901	0.0918	0.0934	0.0951
-1.3	0.0968	0.0985	0.1003	0.1020	0.1038	0.1056	0.1075	0.1093	0.1112	0.1131
-1.2	0.1151	0.1170	0.1190	0.1210	0.1230	0.1251	0.1271	0.1292	0.1314	0.1335
-1.1	0.1357	0.1379	0.1401	0.1423	0.1446	0.1469	0.1492	0.1515	0.1539	0.1562
-1.0	0.1587	0.1611	0.1635	0.1660	0.1685	0.1711	0.1736	0.1762	0.1788	0.1814
-0.9	0.1841	0.1867	0.1894	0.1922	0.1949	0.1977	0.2005	0.2033	0.2061	0.2090
-0.8	0.2119	0.2148	0.2177	0.2206	0.2236	0.2266	0.2296	0.2327	0.2358	0.2389
-0.7	0.2420	0.2451	0.2483	0.2514	0.2546	0.2578	0.2611	0.2643	0.2676	0.2709
-0.6	0.2743	0.2776	0.2810	0.2843	0.2877	0.2912	0.2946	0.2981	0.3015	0.3050
-0.5	0.3085	0.3121	0.3156	0.3192	0.3228	0.3264	0.3300	0.3336	0.3372	0.3409
-0.4	0.3446	0.3483	0.3520	0.3557	0.3594	0.3632	0.3669	0.3707	0.3745	0.3783
-0.3	0.3821	0.3859	0.3897	0.3936	0.3974	0.4013	0.4052	0.4090	0.4129	0.4168
-0.2	0.4207	0.4247	0.4286	0.4325	0.4364	0.4404	0.4443	0.4483	0.4522	0.4562
-0.1	0.4602	0.4641	0.4681	0.4721	0.4761	0.4801	0.4840	0.4880	0.4920	0.4960
0.0	0.5000	0.5040	0.5080	0.5120	0.5160	0.5199	0.5239	0.5279	0.5319	0.5359
0.1	0.5398	0.5438	0.5478	0.5517	0.5557	0.5596	0.5636	0.5675	0.5714	0.5753
0.2	0.5793	0.5832	0.5871	0.5910	0.5948	0.5987	0.6026	0.6064	0.6103	0.6141
0.3	0.6179	0.6217	0.6255	0.6293	0.6331	0.6368	0.6406	0.6443	0.6480	0.6517
0.4	0.6554	0.6591	0.6628	0.6664	0.6700	0.6736	0.6772	0.6808	0.6844	0.6879
0.5	0.6915	0.6950	0.6985	0.7019	0.7054	0.7088	0.7123	0.7157	0.7190	0.7224
0.6	0.7257	0.7291	0.7324	0.7357	0.7389	0.7422	0.7454	0.7486	0.7517	0.7549

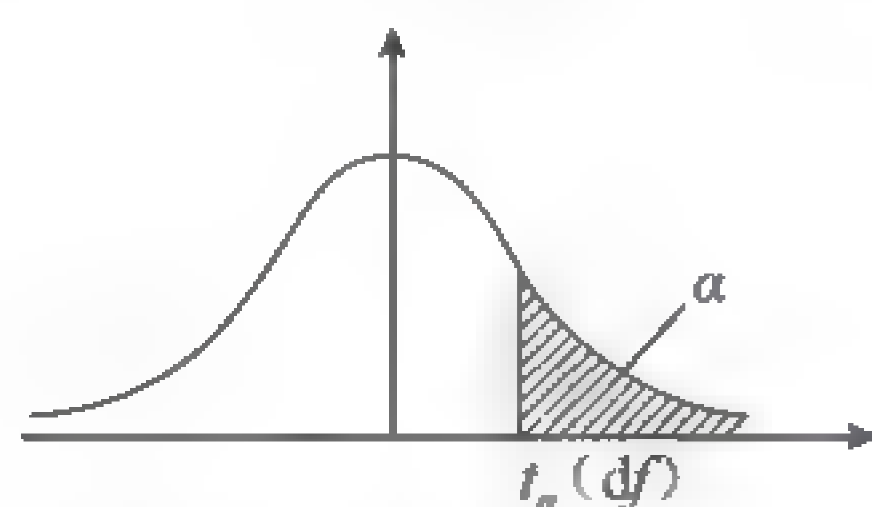
（续表）

z	J									
	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
0.7	0.7580	0.7611	0.7642	0.7673	0.7704	0.7734	0.7764	0.7794	0.7823	0.7852
0.8	0.7881	0.7910	0.7939	0.7967	0.7995	0.8023	0.8051	0.8078	0.8106	0.8133
0.9	0.8159	0.8186	0.8212	0.8238	0.8264	0.8289	0.8315	0.8340	0.8365	0.8389
1.0	0.8413	0.8438	0.8461	0.8485	0.8508	0.8531	0.8554	0.8577	0.8599	0.8621
1.1	0.8643	0.8665	0.8686	0.8708	0.8729	0.8749	0.8770	0.8790	0.8810	0.8830
1.2	0.8849	0.8869	0.8888	0.8907	0.8925	0.8944	0.8962	0.8980	0.8997	0.9015
1.3	0.9032	0.9049	0.9066	0.9082	0.9099	0.9115	0.9131	0.9147	0.9162	0.9177
1.4	0.9192	0.9207	0.9222	0.9236	0.9251	0.9265	0.9279	0.9292	0.9306	0.9319
1.5	0.9332	0.9345	0.9357	0.9370	0.9382	0.9394	0.9406	0.9418	0.9429	0.9441
1.6	0.9452	0.9463	0.9474	0.9484	0.9495	0.9505	0.9515	0.9525	0.9535	0.9545
1.7	0.9554	0.9564	0.9573	0.9582	0.9591	0.9599	0.9608	0.9616	0.9625	0.9633
1.8	0.9641	0.9649	0.9656	0.9664	0.9671	0.9678	0.9686	0.9693	0.9699	0.9706
1.9	0.9713	0.9719	0.9726	0.9732	0.9738	0.9744	0.9750	0.9756	0.9761	0.9767
2.0	0.9772	0.9778	0.9783	0.9788	0.9793	0.9798	0.9803	0.9808	0.9812	0.9817
2.1	0.9821	0.9826	0.9830	0.9834	0.9838	0.9842	0.9846	0.9850	0.9854	0.9857
2.2	0.9861	0.9864	0.9868	0.9871	0.9875	0.9878	0.9881	0.9884	0.9887	0.9890
2.3	0.9893	0.9896	0.9898	0.9901	0.9904	0.9906	0.9909	0.9911	0.9913	0.9916
2.4	0.9918	0.9920	0.9922	0.9925	0.9927	0.9929	0.9931	0.9932	0.9934	0.9936
2.5	0.9938	0.9940	0.9941	0.9943	0.9945	0.9946	0.9948	0.9949	0.9951	0.9952
2.6	0.9953	0.9955	0.9956	0.9957	0.9959	0.9960	0.9961	0.9962	0.9963	0.9964
2.7	0.9965	0.9966	0.9967	0.9968	0.9969	0.9970	0.9971	0.9972	0.9973	0.9974
2.8	0.9974	0.9975	0.9976	0.9977	0.9977	0.9978	0.9979	0.9979	0.9980	0.9981
2.9	0.9981	0.9982	0.9982	0.9983	0.9984	0.9984	0.9985	0.9985	0.9986	0.9986
3.0	0.9987	0.9987	0.9987	0.9988	0.9988	0.9989	0.9989	0.9989	0.9990	0.9990
3.1	0.9990	0.9991	0.9991	0.9991	0.9992	0.9992	0.9992	0.9992	0.9993	0.9993
3.2	0.9993	0.9993	0.9994	0.9994	0.9994	0.9994	0.9994	0.9995	0.9995	0.9995
3.3	0.9995	0.9995	0.9995	0.9996	0.9996	0.9996	0.9996	0.9996	0.9996	0.9997
3.4	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9998
3.5	0.9998	0.9998	0.9998	0.9998	0.9998	0.9998	0.9998	0.9998	0.9998	0.9998

附 录 4

t 分布临界值表，查表参数为显著性水平 α 和自由度参数 df

$$P\left[t(df)>t_\alpha(df)\right]=\alpha$$



程序28-4　生成 t 分布表：自由度 df，显著性水平 α

```
data QuanT;
  array a[6] (0.250, 0.100, 0.050, 0.025, 0.010, 0.005);
  do df=1 to 120;
    array c[6];
    do i= 1 to dim(a);
      p=1-a[i];          /* a[i] 为置信区间的显著性水平 Alpha*/
      c[i]=tinv(p, df); /* 临界值 ta, 即第 p 个分位数*/
    end;
    format c: 9.4;
    keep df c:;
    output;
  end;
run;
/* 创建列标签 0.250 - 0.005 并打印*/
%macro SetLabel();
  data QuanT;
    set QuanT;
    label c1="0.250" c2="0.100" c3="0.050"
          c4="0.025" c5="0.010" c6="0.005";
  run;
%mend;
%SetLabel();
proc print label;run;
```

运行程序输出如下 t 分布临界值 $t_\alpha(df)$ 表，第 1 列为自由度 df，第 1 行为显著性水平 α。单元格数值为置信水平 1 α 所对应的分位数。当显著性水平为 0.250, 0.100, 0.050, 0.025, 0.010, 0.005，对应置信水平为 75%, 90%, 95%, 97.5%, 99%, 99.5%。

服从正态分布但总体方差未知时，t 检验可用于单个总体均值的检验。t 检验也可用于独立小样本的两个总体之均值进行检验。在对两个样本均值是否存在显著性差异时需要使用 t 检验，其原理就是根据样本计算出 t 值与 t 分布临界值表中查到的临界值进行比较。当自由度 df 增大时，t 分布逐渐趋近于正态分布。

表 28-4 t 分布临界值表

df	α					
	0.250	0.100	0.050	0.025	0.010	0.005
1	1.0000	3.0777	6.3138	12 7062	31.8205	63.6567
2	0 8165	1.8856	2.9200	4.3027	6.9646	9.9248
3	0.7649	1.6377	2.3534	3.1824	4.5407	5.8409
4	0 7407	1.5332	2.1318	2 7764	3.7469	4.6041
5	0 7267	1.4759	2.0150	2.5706	3.3649	4.0321
6	0 7176	1.4398	1.9432	2.4469	3.1427	3.7074
7	0 7111	1.4149	1.8946	2.3646	2.9980	3.4995
8	0 7064	1.3968	1.8595	2.3060	2.8965	3.3554
9	0.7027	1.3830	1.8331	2.2622	2.8214	3.2498
10	0 6998	1.3722	1.8125	2.2281	2.7638	3.1693
11	0 6974	1.3634	1.7959	2.2010	2.7181	3.1058
12	0.6955	1.3562	1.7823	2 1788	2.6810	3.0545
13	0 6938	1.3502	1.7709	2.1604	2.6503	3.0123
14	0.6924	1.3450	1.7613	2 1448	2.6245	2.9768
15	0.6912	1.3406	1.7531	2.1314	2.6025	2.9467
16	0.6901	1.3368	1.7459	2.1199	2.5835	2.9208
17	0.6892	1.3334	1.7396	2.1098	2.5669	2.8982
18	0.6884	1.3304	1.7341	2.1009	2.5524	2.8784
19	0.6876	1.3277	1.7291	2.0930	2.5395	2.8609
20	0.6870	1.3253	1.7247	2.0860	2.5280	2.8453
21	0.6864	1.3232	1.7207	2.0796	2.5176	2.8314
22	0.6858	1.3212	1.7171	2.0739	2.5083	2.8188
23	0.6853	1.3195	1.7139	2.0687	2.4999	2.8073
24	0.6848	1.3178	1.7109	2.0639	2.4922	2.7969
25	0.6844	1.3163	1.7081	2.0595	2.4851	2.7874
26	0.6840	1.3150	1.7056	2.0555	2.4786	2.7787
27	0 6837	1.3137	1.7033	2.0518	2.4727	2.7707
28	0.6834	1.3125	1.7011	2.0484	2.4671	2.7633
29	0 6830	1.3114	1.6991	2.0452	2.4620	2.7564
30	0.6828	1.3104	1.6973	2 0423	2 4573	2.7500
31	0 6825	1.3095	1.6955	2.0395	2.4528	2.7440
32	0 6822	1.3086	1.6939	2.0369	2.4487	2.7385
33	0 6820	1.3077	1.6924	2 0345	2.4448	2.7333
34	0.6818	1.3070	1.6909	2.0322	2.4411	2.7284
35	0 6816	1.3062	1.6896	2.0301	2.4377	2.7238
36	0.6814	1.3055	1.6883	2.0281	2.4345	2.7195
37	0 6812	1.3049	1.6871	2.0262	2.4314	2.7154
38	0 6810	1.3042	1.6860	2.0244	2.4286	2.7116
39	0 6808	1.3036	1.6849	2.0227	2.4258	2.7079
40	0.6807	1.3031	1.6839	2.0211	2.4233	2.7045
41	0 6805	1.3025	1.6829	2.0195	2.4208	2.7012
42	0.6804	1.3020	1.6820	2.0181	2.4185	2.6981

（续表）

df	α					
	0.250	0.100	0.050	0.025	0.010	0.005
43	0.6802	1.3016	1.6811	2.0167	2.4163	2.6951
44	0.6801	1.3011	1.6802	2.0154	2.4141	2.6923
45	0.6800	1.3006	1.6794	2.0141	2.4121	2.6896
46	0.6799	1.3002	1.6787	2.0129	2.4102	2.6870
47	0.6797	1.2998	1.6779	2.0117	2.4083	2.6846
48	0.6796	1.2994	1.6772	2.0106	2.4066	2.6822
49	0.6795	1.2991	1.6766	2.0096	2.4049	2.6800
50	0.6794	1.2987	1.6759	2.0086	2.4033	2.6778
51	0.6793	1.2984	1.6753	2.0076	2.4017	2.6757
52	0.6792	1.2980	1.6747	2.0066	2.4002	2.6737
53	0.6791	1.2977	1.6741	2.0057	2.3988	2.6718
54	0.6791	1.2974	1.6736	2.0049	2.3974	2.6700
55	0.6790	1.2971	1.6730	2.0040	2.3961	2.6682
56	0.6789	1.2969	1.6725	2.0032	2.3948	2.6665
57	0.6788	1.2966	1.6720	2.0025	2.3936	2.6649
58	0.6787	1.2963	1.6716	2.0017	2.3924	2.6633
59	0.6787	1.2961	1.6711	2.0010	2.3912	2.6618
60	0.6786	1.2958	1.6706	2.0003	2.3901	2.6603
61	0.6785	1.2956	1.6702	1.9996	2.3890	2.6589
62	0.6785	1.2954	1.6698	1.9990	2.3880	2.6575
63	0.6784	1.2951	1.6694	1.9983	2.3870	2.6561
64	0.6783	1.2949	1.6690	1.9977	2.3860	2.6549
65	0.6783	1.2947	1.6686	1.9971	2.3851	2.6536
66	0.6782	1.2945	1.6683	1.9966	2.3842	2.6524
67	0.6782	1.2943	1.6679	1.9960	2.3833	2.6512
68	0.6781	1.2941	1.6676	1.9955	2.3824	2.6501
69	0.6781	1.2939	1.6672	1.9949	2.3816	2.6490
70	0.6780	1.2938	1.6669	1.9944	2.3808	2.6479
71	0.6780	1.2936	1.6666	1.9939	2.3800	2.6469
72	0.6779	1.2934	1.6663	1.9935	2.3793	2.6459
73	0.6779	1.2933	1.6660	1.9930	2.3785	2.6449
74	0.6778	1.2931	1.6657	1.9925	2.3778	2.6439
75	0.6778	1.2929	1.6654	1.9921	2.3771	2.6430
76	0.6777	1.2928	1.6652	1.9917	2.3764	2.6421
77	0.6777	1.2926	1.6649	1.9913	2.3758	2.6412
78	0.6776	1.2925	1.6646	1.9908	2.3751	2.6403
79	0.6776	1.2924	1.6644	1.9905	2.3745	2.6395
80	0.6776	1.2922	1.6641	1.9901	2.3739	2.6387
81	0.6775	1.2921	1.6639	1.9897	2.3733	2.6379
82	0.6775	1.2920	1.6636	1.9893	2.3727	2.6371
83	0.6775	1.2918	1.6634	1.9890	2.3721	2.6364
84	0.6774	1.2917	1.6632	1.9886	2.3716	2.6356

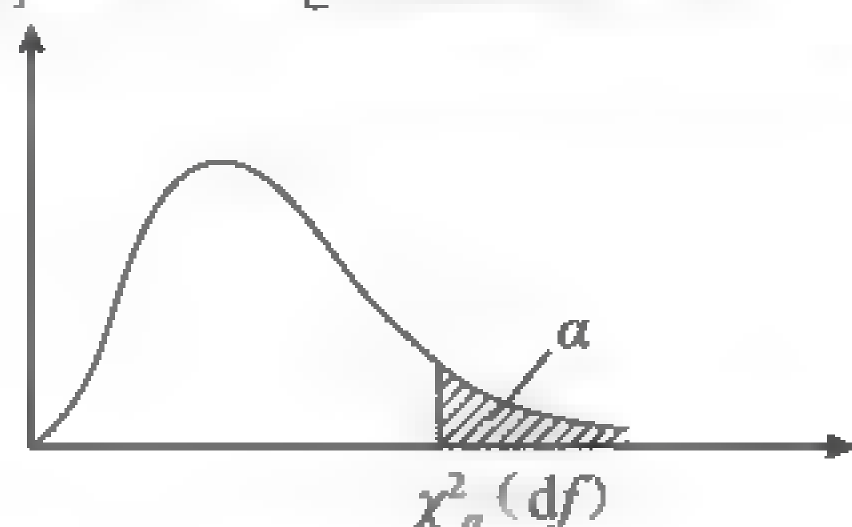
（续表）

df	α					
	0.250	0.100	0.050	0.025	0.010	0.005
85	0.6774	1.2916	1.6630	1.9883	2.3710	2.6349
86	0.6774	1.2915	1.6628	1.9879	2.3705	2.6342
87	0.6773	1.2914	1.6626	1.9876	2.3700	2.6335
88	0.6773	1.2912	1.6624	1.9873	2.3695	2.6329
89	0.6773	1.2911	1.6622	1.9870	2.3690	2.6322
90	0.6772	1.2910	1.6620	1.9867	2.3685	2.6316
91	0.6772	1.2909	1.6618	1.9864	2.3680	2.6309
92	0.6772	1.2908	1.6616	1.9861	2.3676	2.6303
93	0.6771	1.2907	1.6614	1.9858	2.3671	2.6297
94	0.6771	1.2906	1.6612	1.9855	2.3667	2.6291
95	0.6771	1.2905	1.6611	1.9853	2.3662	2.6286
96	0.6771	1.2904	1.6609	1.9850	2.3658	2.6280
97	0.6770	1.2903	1.6607	1.9847	2.3654	2.6275
98	0.6770	1.2902	1.6606	1.9845	2.3650	2.6269
99	0.6770	1.2902	1.6604	1.9842	2.3646	2.6264
100	0.6770	1.2901	1.6602	1.9840	2.3642	2.6259
101	0.6769	1.2900	1.6601	1.9837	2.3638	2.6254
102	0.6769	1.2899	1.6599	1.9835	2.3635	2.6249
103	0.6769	1.2898	1.6598	1.9833	2.3631	2.6244
104	0.6769	1.2897	1.6596	1.9830	2.3627	2.6239
105	0.6768	1.2897	1.6595	1.9828	2.3624	2.6235
106	0.6768	1.2896	1.6594	1.9826	2.3620	2.6230
107	0.6768	1.2895	1.6592	1.9824	2.3617	2.6226
108	0.6768	1.2894	1.6591	1.9822	2.3614	2.6221
109	0.6767	1.2894	1.6590	1.9820	2.3610	2.6217
110	0.6767	1.2893	1.6588	1.9818	2.3607	2.6213
111	0.6767	1.2892	1.6587	1.9816	2.3604	2.6208
112	0.6767	1.2892	1.6586	1.9814	2.3601	2.6204
113	0.6767	1.2891	1.6585	1.9812	2.3598	2.6200
114	0.6766	1.2890	1.6583	1.9810	2.3595	2.6196
115	0.6766	1.2890	1.6582	1.9808	2.3592	2.6193
116	0.6766	1.2889	1.6581	1.9806	2.3589	2.6189
117	0.6766	1.2888	1.6580	1.9804	2.3586	2.6185
118	0.6766	1.2888	1.6579	1.9803	2.3584	2.6181
119	0.6766	1.2887	1.6578	1.9801	2.3581	2.6178
120	0.6765	1.2886	1.6577	1.9799	2.3578	2.6174
∞	0.6745	1.2816	1.6449	1.9600	2.3263	2.5758

附 录 5

χ^2 分布临界值表，查表参数为显著性水平 α 和自由度参数 df

$$P\left[\chi^2(df)>\chi^2_\alpha(df)\right]=\alpha \qquad P\left[\chi^2_{1-\alpha/2}(df)\leqslant\chi^2(df)\leqslant\chi^2_{\alpha/2}(df)\right]=1-\alpha$$



程序28-5 生成卡方分布临界值表：自由度df，显著性水平α

```
data QuanC;
  array a[10] (0.995, 0.990, 0.975, 0.950, 0.900,
               0.100, 0.050, 0.025, 0.010, 0.005);
  do df=1 to 120;
    array c[10];
    do i= 1 to dim(a);
      p=1-a[i];          /* a[i] 为置信区间的显著性水平 Alpha*/
      c[i]= cinv(p, df); /* 临界值 Xa, 即第 p 个分位数*/
    end;
    format c: 9.4;
    keep df c: ;
    output;
  end;
run;
/* 创建列标签 0.995 - 0.005 并打印*/
%macro SetLabel();
  data QuanC;
    set QuanC;
    label c1="0.995" c2="0.990" c3="0.975" c4="0.950" c5="0.900"
          c6="0.100" c7="0.050" c8="0.025" c9="0.010" c10="0.005";
  run;
%mend;
%SetLabel();
proc print label;run;
```

运行程序输出如下 χ^2 分布临界值 $\chi^2_\alpha(df)$ 表，第 1 列为自由度 df，第 1 行为显著性水平 α。单元格数值为置信水平 $1-\alpha$ 所对应的分位数。当显著性水平为 0.050 和 0.010 时，对应置信水平为 95% 和 99%。

服从正态分布的随机变量的平方和构成的随机变量服从 χ^2 分布。参数检验中对一个总体方差进行点估计时构造的卡方统计量服从 χ^2 分布。卡方检验可验证两个总体某个比率之间是否存在显著性差异，卡方值越大反映相互偏离程度越高。

在 SAS 中分布的临界值也可通过分位数函数 QUANTILE 函数求得，比如 χ^2 分布临界值 `CriticalValue = QUANTILE('CHISQ',1-alpha,df);` 假设检验中用如下方法计算随机变量 x 所对应的 P- 值：`PValue = 1 - PROBCHI(x, df);` 以判断接纳或拒绝原假设。

表 28-5 χ^2 分布临界值表

df	α=0.995	0.990	0.975	0.950	0.900	0.100	0.050	0.025	0.010	0.005
1	0.0000	0.0002	0.0010	0.0039	0.0158	2.7055	3.8415	5.0239	6.6349	7.8794
2	0.0100	0.0201	0.0506	0.1026	0.2107	4.6052	5.9915	7.3778	9.2103	10.5966
3	0.0717	0.1148	0.2158	0.3518	0.5844	6.2514	7.8147	9.3484	11.3449	12.8382
4	0.2070	0.2971	0.4844	0.7107	1.0636	7.7794	9.4877	11.1433	13.2767	14.8603
5	0.4117	0.5543	0.8312	1.1455	1.6103	9.2364	11.0705	12.8325	15.0863	16.7496
6	0.6757	0.8721	1.2373	1.6354	2.2041	10.6446	12.5916	14.4494	16.8119	18.5476
7	0.9893	1.2390	1.6899	2.1673	2.8331	12.0170	14.0671	16.0128	18.4753	20.2777
8	1.3444	1.6465	2.1797	2.7326	3.4895	13.3616	15.5073	17.5345	20.0902	21.9550
9	1.7349	2.0879	2.7004	3.3251	4.1682	14.6837	16.9190	19.0228	21.6660	23.5894
10	2.1559	2.5582	3.2470	3.9403	4.8652	15.9872	18.3070	20.4832	23.2093	25.1882
11	2.6032	3.0535	3.8157	4.5748	5.5778	17.2750	19.6751	21.9200	24.7250	26.7568
12	3.0738	3.5706	4.4038	5.2260	6.3038	18.5493	21.0261	23.3367	26.2170	28.2995
13	3.5650	4.1069	5.0088	5.8919	7.0415	19.8119	22.3620	24.7356	27.6882	29.8195
14	4.0747	4.6604	5.6287	6.5706	7.7895	21.0641	23.6848	26.1189	29.1412	31.3193
15	4.6009	5.2293	6.2621	7.2609	8.5468	22.3071	24.9958	27.4884	30.5779	32.8013
16	5.1422	5.8122	6.9077	7.9616	9.3122	23.5418	26.2962	28.8454	31.9999	34.2672
17	5.6972	6.4078	7.5642	8.6718	10.0852	24.7690	27.5871	30.1910	33.4087	35.7185
18	6.2648	7.0149	8.2307	9.3905	10.8649	25.9894	28.8693	31.5264	34.8053	37.1565
19	6.8440	7.6327	8.9065	10.1170	11.6509	27.2036	30.1435	32.8523	36.1909	38.5823
20	7.4338	8.2604	9.5908	10.8508	12.4426	28.4120	31.4104	34.1696	37.5662	39.9968
21	8.0337	8.8972	10.2829	11.5913	13.2396	29.6151	32.6706	35.4789	38.9322	41.4011
22	8.6427	9.5425	10.9823	12.3380	14.0415	30.8133	33.9244	36.7807	40.2894	42.7957
23	9.2604	10.1957	11.6886	13.0905	14.8480	32.0069	35.1725	38.0756	41.6384	44.1813
24	9.8862	10.8564	12.4012	13.8484	15.6587	33.1962	36.4150	39.3641	42.9798	45.5585
25	10.5197	11.5240	13.1197	14.6114	16.4734	34.3816	37.6525	40.6465	44.3141	46.9279
26	11.1602	12.1981	13.8439	15.3792	17.2919	35.5632	38.8851	41.9232	45.6417	48.2899
27	11.8076	12.8785	14.5734	16.1514	18.1139	36.7412	40.1133	43.1945	46.9629	49.6449
28	12.4613	13.5647	15.3079	16.9279	18.9392	37.9159	41.3371	44.4608	48.2782	50.9934
29	13.1211	14.2565	16.0471	17.7084	19.7677	39.0875	42.5570	45.7223	49.5879	52.3356
30	13.7867	14.9535	16.7908	18.4927	20.5992	40.2560	43.7730	46.9792	50.8922	53.6720
31	14.4578	15.6555	17.5387	19.2806	21.4336	41.4217	44.9853	48.2319	52.1914	55.0027
32	15.1340	16.3622	18.2908	20.0719	22.2706	42.5847	46.1943	49.4804	53.4858	56.3281
33	15.8153	17.0735	19.0467	20.8665	23.1102	43.7452	47.3999	50.7251	54.7755	57.6484
34	16.5013	17.7891	19.8063	21.6643	23.9523	44.9032	48.6024	51.9660	56.0609	58.9639
35	17.1918	18.5089	20.5694	22.4650	24.7967	46.0588	49.8018	53.2033	57.3421	60.2748
36	17.8867	19.2327	21.3359	23.2686	25.6433	47.2122	50.9985	54.4373	58.6192	61.5812
37	18.5858	19.9602	22.1056	24.0749	26.4921	48.3634	52.1923	55.6680	59.8925	62.8833
38	19.2889	20.6914	22.8785	24.8839	27.3430	49.5126	53.3835	56.8955	61.1621	64.1814
39	19.9959	21.4262	23.6543	25.6954	28.1958	50.6598	54.5722	58.1201	62.4281	65.4756
40	20.7065	22.1643	24.4330	26.5093	29.0505	51.8051	55.7585	59.3417	63.6907	66.7660
41	21.4208	22.9056	25.2145	27.3256	29.9071	52.9485	56.9424	60.5606	64.9501	68.0527
42	22.1385	23.6501	25.9987	28.1440	30.7654	54.0902	58.1240	61.7768	66.2062	69.3360
43	22.8595	24.3976	26.7854	28.9647	31.6255	55.2302	59.3035	62.9904	67.4593	70.6159

（续表）

df	α=0.995	0.990	0.975	0.950	0.900	0.100	0.050	0.025	0.010	0.005
44	23.5837	25.1480	27.5746	29.7875	32.4871	56.3685	60.4809	64.2015	68.7095	71.8926
45	24.3110	25.9013	28.3662	30.6123	33.3504	57.5053	61.6562	65.4102	69.9568	73.1661
46	25.0413	26.6572	29.1601	31.4390	34.2152	58.6405	62.8296	66.6165	71.2014	74.4365
47	25.7746	27.4158	29.9562	32.2676	35.0814	59.7743	64.0011	67.8206	72.4433	75.7041
48	26.5106	28.1770	30.7545	33.0981	35.9491	60.9066	65.1708	69.0226	73.6826	76.9688
49	27.2493	28.9406	31.5549	33.9303	36.8182	62.0375	66.3386	70.2224	74.9195	78.2307
50	27.9907	29.7067	32.3574	34.7643	37.6886	63.1671	67.5048	71.4202	76.1539	79.4900
51	28.7347	30.4750	33.1618	35.5999	38.5604	64.2954	68.6693	72.6160	77.3860	80.7467
52	29.4812	31.2457	33.9681	36.4371	39.4334	65.4224	69.8322	73.8099	78.6158	82.0008
53	30.2300	32.0185	34.7763	37.2759	40.3076	66.5482	70.9935	75.0019	79.8433	83.2526
54	30.9813	32.7934	35.5863	38.1162	41.1830	67.6728	72.1532	76.1920	81.0688	84.5019
55	31.7348	33.5705	36.3981	38.9580	42.0596	68.7962	73.3115	77.3805	82.2921	85.7490
56	32.4905	34.3495	37.2116	39.8013	42.9373	69.9185	74.4683	78.5672	83.5134	86.9938
57	33.2484	35.1305	38.0267	40.6459	43.8161	71.0397	75.6237	79.7522	84.7328	88.2364
58	34.0084	35.9135	38.8435	41.4920	44.6960	72.1598	76.7778	80.9356	85.9502	89.4769
59	34.7704	36.6982	39.6619	42.3393	45.5770	73.2789	77.9305	82.1174	87.1657	90.7153
60	35.5345	37.4849	40.4817	43.1880	46.4589	74.3970	79.0819	83.2977	88.3794	91.9517
61	36.3005	38.2732	41.3031	44.0379	47.3418	75.5141	80.2321	84.4764	89.5913	93.1861
62	37.0684	39.0633	42.1260	44.8890	48.2257	76.6302	81.3810	85.6537	90.8015	94.4187
63	37.8382	39.8551	42.9503	45.7414	49.1105	77.7454	82.5287	86.8296	92.0100	95.6493
64	38.6098	40.6486	43.7760	46.5949	49.9963	78.8596	83.6753	88.0041	93.2169	96.8781
65	39.3831	41.4436	44.6030	47.4496	50.8829	79.9730	84.8206	89.1771	94.4221	98.1051
66	40.1582	42.2402	45.4314	48.3054	51.7705	81.0855	85.9649	90.3489	95.6257	99.3304
67	40.9350	43.0384	46.2610	49.1623	52.6588	82.1971	87.1081	91.5194	96.8278	100.5540
68	41.7135	43.8380	47.0920	50.0202	53.5481	83.3079	88.2502	92.6885	98.0284	101.7759
69	42.4935	44.6392	47.9242	50.8792	54.4381	84.4179	89.3912	93.8565	99.2275	102.9962
70	43.2752	45.4417	48.7576	51.7393	55.3289	85.5270	90.5312	95.0232	100.4252	104.2149
71	44.0584	46.2457	49.5922	52.6003	56.2206	86.6354	91.6702	96.1887	101.6214	105.4320
72	44.8431	47.0510	50.4279	53.4623	57.1129	87.7430	92.8083	97.3531	102.8163	106.6476
73	45.6293	47.8577	51.2648	54.3253	58.0061	88.8499	93.9453	98.5163	104.0098	107.8617
74	46.4170	48.6657	52.1028	55.1892	58.9000	89.9560	95.0815	99.6783	105.2020	109.0744
75	47.2060	49.4750	52.9419	56.0541	59.7946	91.0615	96.2167	100.8393	106.3929	110.2856
76	47.9965	50.2856	53.7821	56.9198	60.6899	92.1662	97.3510	101.9993	107.5825	111.4954
77	48.7884	51.0974	54.6234	57.7864	61.5858	93.2702	98.4844	103.1581	108.7709	112.7038
78	49.5816	51.9104	55.4656	58.6539	62.4825	94.3735	99.6169	104.3159	109.9581	113.9109
79	50.3761	52.7247	56.3089	59.5223	63.3799	95.4762	100.7486	105.4728	111.1440	115.1166
80	51.1719	53.5401	57.1532	60.3915	64.2778	96.5782	101.8795	106.6286	112.3288	116.3211
81	51.9690	54.3566	57.9984	61.2615	65.1765	97.6796	103.0095	107.7834	113.5124	117.5242
82	52.7674	55.1743	58.8446	62.1323	66.0757	98.7803	104.1387	108.9373	114.6949	118.7261
83	53.5669	55.9931	59.6918	63.0039	66.9756	99.8805	105.2672	110.0902	115.8763	119.9268
84	54.3677	56.8130	60.5398	63.8763	67.8761	100.9800	106.3948	111.2423	117.0565	121.1263
85	55.1696	57.6339	61.3888	64.7494	68.7772	102.0789	107.5217	112.3934	118.2357	122.3246
86	55.9727	58.4559	62.2386	65.6233	69.6788	103.1773	108.6479	113.5436	119.4139	123.5217

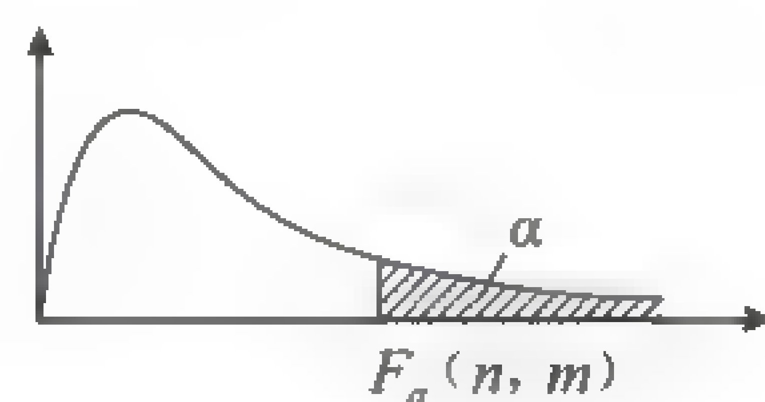
（续表）

df	α=0.995	0.990	0.975	0.950	0.900	0.100	0.050	0.025	0.010	0 005
87	56.7769	59.2790	63.0894	66.4979	70.5810	104.2750	109.7733	114.6929	120.5910	124.7177
88	57.5823	60.1030	63.9409	67.3732	71.4838	105.3722	110 8980	115.8414	121.7671	125.9125
89	58.3888	60.9281	64.7934	68.2493	72.3872	106.4689	112.0220	116.9891	122.9422	127.1063
90	59.1963	61.7541	65.6466	69.1260	73.2911	107.5650	113.1453	118.1359	124.1163	128.2989
91	60.0049	62.5811	66.5007	70.0035	74.1955	108.6606	114.2679	119.2819	125.2895	129.4905
92	60 8146	63.4090	67.3556	70.8816	75.1005	109.7556	115.3898	120.4271	126.4617	130.6811
93	61.6253	64.2379	68.2112	71.7603	76.0060	110 8502	116.5110	121.5715	127.6329	131.8706
94	62.4370	65.0677	69.0677	72.6398	76.9119	111.9442	117.6317	122.7151	128.8032	133.0591
95	63.2496	65.8984	69.9249	73.5198	77.8184	113.0377	118.7516	123.8580	129.9727	134.2465
96	64.0633	66.7299	70.7828	74.4005	78.7254	114.1307	119.8709	125.0001	131.1412	135.4330
97	64.8780	67.5624	71.6415	75.2819	79.6329	115.2232	120.9896	126.1414	132.3089	136.6186
98	65.6936	68.3957	72.5009	76.1638	80.5408	116.3153	122.1077	127.2821	133.4757	137.8032
99	66.5101	69.2299	73.3611	77.0463	81.4493	117.4069	123.2252	128.4220	134.6416	138.9868
100	67.3276	70.0649	74.2219	77.9295	82.3581	118.4980	124.3421	129.5612	135.8067	140.1695
101	68.1459	70.9007	75.0835	78 8132	83.2675	119.5887	125.4584	130.6997	136.9710	141.3513
102	68.9652	71.7374	75.9457	79.6975	84.1773	120.6789	126.5741	131.8375	138.1345	142.5322
103	69.7853	72.5748	76.8086	80.5823	85.0875	121.7686	127.6893	132.9747	139.2971	143.7122
104	70.6064	73.4130	77.6722	81.4678	85.9982	122.8580	128.8039	134.1112	140.4590	144.8913
105	71.4282	74.2520	78.5364	82.3537	86.9093	123.9469	129.9180	135.2470	141.6201	146.0696
106	72.2509	75.0918	79.4013	83 2403	87.8208	125.0354	131.0315	136.3822	142.7804	147.2470
107	73.0745	75.9323	80.2668	84.1273	88.7327	126.1234	132.1444	137.5167	143.9400	148.4236
108	73.8989	76.7736	81.1329	85.0149	89.6451	127.2111	133.2569	138.6506	145.0988	149.5994
109	74.7241	77.6156	81.9997	85.9030	90.5579	128.2983	134.3688	139.7839	146.2569	150.7743
110	75.5500	78.4583	82.8671	86.7916	91.4710	129.3851	135.4802	140.9166	147.4143	151.9485
111	76.3768	79.3017	83.7350	87.6808	92.3846	130.4716	136.5911	142.0486	148.5710	153.1218
112	77.2044	80.1459	84.6036	88.5704	93.2986	131.5576	137.7015	143.1801	149.7269	154.2944
113	78.0327	80.9907	85.4728	89.4605	94.2129	132.6433	138.8114	144.3110	150.8822	155.4662
114	78.8618	81.8362	86.3425	90.3511	95.1276	133.7286	139.9208	145.4413	152.0367	156.6373
115	79.6916	82.6824	87.2128	91.2422	96.0427	134.8135	141.0297	146.5711	153.1906	157.8076
116	80.5221	83.5293	88.0837	92.1338	96.9582	135.8980	142.1382	147.7002	154.3438	158.9771
117	81.3534	84.3768	88.9551	93.0258	97.8740	136.9822	143.2461	148.8288	155.4964	160.1460
118	82.1854	85.2250	89.8271	93.9183	98.7902	138.0660	144.3537	149.9569	156.6483	161.3141
119	83 0182	86.0738	90.6996	94.8112	99.7067	139.1495	145.4607	151.0844	157.7995	162.4815
120	83.8516	86.9233	91.5726	95.7046	100 6236	140.2326	146.5674	152.2114	158.9502	163.6482

附 录 6

F 分布临界值表，显著性水平 α 和两个自由度参数 n 和 m

$$P[F(n,m)] > F_\alpha(n,m) = \alpha$$



程序28-6 生成 F 分布临界值表：$F(n,m,\alpha)$

```
data QuanF;
  array ndf[20] _temporary_ (1,  2,  3,  4,   5,  6,  7,  8,   9,  10,
                            12, 15, 20, 24,  30, 40, 50, 60, 100, 120);
  array ddf[43] _temporary_;
  do i=1 to 30;
   ddf[i]=i;
  end;
  do i=40 to 120 by 10;
    ddf[30+(i-30)/10]=i;
  end;
  do i=140 to 200 by 20;
    ddf[39+(i-120)/20]=i;
  end;
  a= 0.10; /* 显著性水平: 0.10, 0.05, 0.025, 0.010, 0.005 */
  do i=1 to dim(ddf);
    m=ddf[i];
    array c[20];
    do j=1 to dim(ndf);
      c[j]=finv( 1- a, ndf[j], ddf[i]);
    end;
    format c: 9.4;
    keep m c: ;
    output;
  end;
run;
/* 创建列标签 1 - 120 并打印*/
%macro SetLabel();
  data QuanF;
    set QuanF;
    label c1="1"   c2="2"   c3="3"   c4="4"   c5="5"    c6="6"    c7="7"
          c8="8"   c9="9"   c10="10" c11="12" c12="15" c13="20"  c14="24"
          c15="30" c16="40" c17="50" c18="60" c19="100" c20="120";
  run;
%mend;
%SetLabel();
proc print label;run;
```

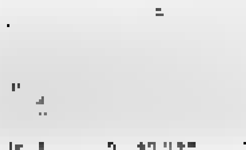


表 28-6　F 分布临界值表 α=0.05

m	n=1	2	3	4	5	6	7	8	9	10	12	15	20	24	30	40	50	60	100	120
1	161.44	199.50	215.70	224.58	230.16	233.98	236.76	238.88	240.54	241.88	243.90	245.94	248.01	249.05	250.09	251.14	251.77	252.19	253.04	253.25
2	18.512	19.000	19.164	19.246	19.296	19.329	19.353	19.371	19.384	19.395	19.412	19.429	19.445	19.454	19.462	19.470	19.475	19.479	19.485	19.487
3	10.128	9.5521	9.2766	9.1172	9.0135	8.9406	8.8867	8.8452	8.8123	8.7855	8.7446	8.7029	8.6602	8.6385	8.6166	8.5944	8.5810	8.5720	8.5539	8.5494
4	7.7086	6.9443	6.5914	6.3882	6.2561	6.1631	6.0942	6.0410	5.9988	5.9644	5.9117	5.8578	5.8025	5.7744	5.7459	5.7170	5.6995	5.6877	5.6641	5.6581
5	6.6079	5.7861	5.4095	5.1922	5.0503	4.9503	4.8759	4.8183	4.7725	4.7351	4.6777	4.6188	4.5581	4.5272	4.4957	4.4638	4.4444	4.4314	4.4051	4.3985
6	5.9874	5.1433	4.7571	4.5337	4.3874	4.2839	4.2067	4.1468	4.0990	4.0600	3.9999	3.9381	3.8742	3.8415	3.8082	3.7743	3.7537	3.7398	3.7117	3.7047
7	5.5914	4.7374	4.3468	4.1203	3.9715	3.8660	3.7870	3.7257	3.6767	3.6365	3.5747	3.5107	3.4445	3.4105	3.3758	3.3404	3.3189	3.3043	3.2749	3.2674
8	5.3177	4.4590	4.0662	3.8379	3.6875	3.5806	3.5005	3.4381	3.3881	3.3472	3.2839	3.2184	3.1503	3.1152	3.0794	3.0428	3.0204	3.0053	2.9747	2.9669
9	5.1174	4.2565	3.8625	3.6331	3.4817	3.3738	3.2927	3.2296	3.1789	3.1373	3.0729	3.0061	2.9365	2.9005	2.8637	2.8259	2.8028	2.7872	2.7556	2.7475
10	4.9646	4.1028	3.7083	3.4780	3.3258	3.2172	3.1355	3.0717	3.0204	2.9782	2.9130	2.8450	2.7740	2.7372	2.6996	2.6609	2.6371	2.6211	2.5884	2.5801
11	4.8443	3.9823	3.5874	3.3567	3.2039	3.0946	3.0123	2.9480	2.8962	2.8536	2.7876	2.7186	2.6464	2.6090	2.5705	2.5309	2.5066	2.4901	2.4566	2.4480
12	4.7472	3.8853	3.4903	3.2592	3.1059	2.9961	2.9134	2.8486	2.7964	2.7534	2.6866	2.6169	2.5436	2.5055	2.4663	2.4259	2.4010	2.3842	2.3498	2.3410
13	4.6672	3.8056	3.4105	3.1791	3.0254	2.9153	2.8321	2.7669	2.7144	2.6710	2.6037	2.5331	2.4589	2.4202	2.3803	2.3392	2.3138	2.2966	2.2614	2.2524
14	4.6001	3.7389	3.3439	3.1122	2.9582	2.8477	2.7642	2.6987	2.6458	2.6022	2.5342	2.4630	2.3879	2.3487	2.3082	2.2664	2.2405	2.2229	2.1870	2.1778
15	4.5431	3.6823	3.2874	3.0556	2.9013	2.7905	2.7066	2.6408	2.5876	2.5437	2.4753	2.4034	2.3275	2.2878	2.2468	2.2043	2.1780	2.1601	2.1234	2.1141
16	4.4940	3.6337	3.2389	3.0069	2.8524	2.7413	2.6572	2.5911	2.5377	2.4935	2.4247	2.3522	2.2756	2.2354	2.1938	2.1507	2.1240	2.1058	2.0685	2.0589
17	4.4513	3.5915	3.1968	2.9647	2.8100	2.6987	2.6143	2.5480	2.4943	2.4499	2.3807	2.3077	2.2304	2.1898	2.1477	2.1040	2.0769	2.0584	2.0204	2.0107
18	4.4139	3.5546	3.1599	2.9277	2.7729	2.6613	2.5767	2.5102	2.4563	2.4117	2.3421	2.2686	2.1906	2.1497	2.1071	2.0629	2.0354	2.0166	1.9780	1.9681
19	4.3807	3.5219	3.1274	2.8951	2.7401	2.6283	2.5435	2.4768	2.4227	2.3779	2.3080	2.2341	2.1555	2.1141	2.0712	2.0264	1.9986	1.9795	1.9403	1.9302
20	4.3512	3.4928	3.0984	2.8661	2.7109	2.5990	2.5140	2.4471	2.3928	2.3479	2.2776	2.2033	2.1242	2.0825	2.0391	1.9938	1.9656	1.9464	1.9066	1.8963
21	4.3248	3.4668	3.0725	2.8401	2.6848	2.5727	2.4876	2.4205	2.3660	2.3210	2.2504	2.1757	2.0960	2.0540	2.0102	1.9645	1.9360	1.9165	1.8761	1.8657
22	4.3009	3.4434	3.0491	2.8167	2.6613	2.5491	2.4638	2.3965	2.3419	2.2967	2.2258	2.1508	2.0707	2.0283	1.9842	1.9380	1.9092	1.8894	1.8486	1.8380
23	4.2793	3.4221	3.0280	2.7955	2.6400	2.5277	2.4422	2.3748	2.3201	2.2747	2.2036	2.1282	2.0476	2.0050	1.9605	1.9139	1.8848	1.8648	1.8234	1.8128

（续表）

m	n=1	2	3	4	5	6	7	8	9	10	12	15	20	24	30	40	50	60	100	120
24	4.2597	3.4028	3.0088	2.7763	2.6207	2.5082	2.4226	2.3551	2.3002	2.2547	2.1834	2.1077	2.0267	1.9838	1.9390	1.8920	1.8625	1.8424	1.8005	1.7896
25	4.2417	3.3852	2.9912	2.7587	2.6030	2.4904	2.4047	2.3371	2.2821	2.2365	2.1649	2.0889	2.0075	1.9643	1.9192	1.8718	1.8421	1.8217	1.7794	1.7684
26	4.2252	3.3690	2.9752	2.7426	2.5868	2.4741	2.3883	2.3205	2.2655	2.2197	2.1479	2.0716	1.9898	1.9464	1.9010	1.8533	1.8233	1.8027	1.7599	1.7488
27	4.2100	3.3541	2.9604	2.7278	2.5719	2.4591	2.3732	2.3053	2.2501	2.2043	2.1323	2.0558	1.9736	1.9299	1.8842	1.8361	1.8059	1.7851	1.7419	1.7306
28	4.1960	3.3404	2.9467	2.7141	2.5581	2.4453	2.3593	2.2913	2.2360	2.1900	2.1179	2.0411	1.9586	1.9147	1.8687	1.8203	1.7898	1.7689	1.7251	1.7138
29	4.1830	3.3277	2.9340	2.7014	2.5454	2.4324	2.3463	2.2783	2.2229	2.1768	2.1045	2.0275	1.9446	1.9005	1.8543	1.8055	1.7748	1.7537	1.7096	1.6981
30	4.1709	3.3158	2.9223	2.6896	2.5336	2.4205	2.3343	2.2662	2.2107	2.1646	2.0921	2.0148	1.9317	1.8874	1.8409	1.7918	1.7609	1.7396	1.6950	1.6835
40	4.0847	3.2317	2.8387	2.6060	2.4495	2.3359	2.2490	2.1802	2.1240	2.0772	2.0035	1.9245	1.8389	1.7929	1.7444	1.6928	1.6600	1.6373	1.5892	1.5766
50	4.0343	3.1826	2.7900	2.5572	2.4004	2.2864	2.1992	2.1299	2.0734	2.0261	1.9515	1.8714	1.7841	1.7371	1.6872	1.6337	1.5995	1.5757	1.5249	1.5115
60	4.0012	3.1504	2.7581	2.5252	2.3683	2.2541	2.1665	2.0970	2.0401	1.9926	1.9174	1.8364	1.7480	1.7001	1.6491	1.5943	1.5590	1.5343	1.4814	1.4673
70	3.9778	3.1277	2.7355	2.5027	2.3456	2.2312	2.1435	2.0737	2.0166	1.9689	1.8932	1.8117	1.7223	1.6738	1.6220	1.5661	1.5300	1.5046	1.4498	1.4351
80	3.9604	3.1108	2.7188	2.4859	2.3287	2.2142	2.1263	2.0564	1.9991	1.9512	1.8753	1.7932	1.7032	1.6542	1.6017	1.5449	1.5081	1.4821	1.4259	1.4107
90	3.9469	3.0977	2.7058	2.4729	2.3157	2.2011	2.1131	2.0430	1.9856	1.9376	1.8613	1.7789	1.6883	1.6389	1.5859	1.5284	1.4910	1.4645	1.4070	1.3914
100	3.9361	3.0873	2.6955	2.4626	2.3053	2.1906	2.1025	2.0323	1.9748	1.9267	1.8503	1.7675	1.6764	1.6267	1.5733	1.5151	1.4772	1.4504	1.3917	1.3757
110	3.9274	3.0788	2.6871	2.4542	2.2969	2.1821	2.0939	2.0236	1.9661	1.9178	1.8412	1.7582	1.6667	1.6167	1.5630	1.5043	1.4660	1.4388	1.3791	1.3628
120	3.9201	3.0718	2.6802	2.4472	2.2899	2.1750	2.0868	2.0164	1.9588	1.9105	1.8337	1.7505	1.6587	1.6084	1.5543	1.4952	1.4565	1.4290	1.3685	1.3519
140	3.9087	3.0608	2.6693	2.4363	2.2789	2.1639	2.0756	2.0051	1.9473	1.8989	1.8219	1.7384	1.6460	1.5954	1.5408	1.4809	1.4416	1.4136	1.3517	1.3345
160	3.9002	3.0525	2.6611	2.4282	2.2707	2.1557	2.0672	1.9967	1.9388	1.8903	1.8131	1.7293	1.6366	1.5856	1.5306	1.4702	1.4304	1.4020	1.3388	1.3213
180	3.8936	3.0461	2.6548	2.4218	2.2643	2.1492	2.0608	1.9901	1.9322	1.8836	1.8063	1.7223	1.6292	1.5780	1.5227	1.4618	1.4217	1.3929	1.3288	1.3109
200	3.8884	3.0411	2.6498	2.4168	2.2592	2.1441	2.0556	1.9849	1.9269	1.8783	1.8008	1.7166	1.6233	1.5720	1.5164	1.4551	1.4146	1.3856	1.3206	1.3024

表 28-7　F 分布临界值表 $\alpha=0.01$

m	n=1	2	3	4	5	6	7	8	9	10	12	15	20	24	30	40	50	60	100	120
1	4052.1	4999.5	5403.3	5624.5	5763.6	5858.9	5928.3	5981.0	6022.4	6055.8	6106.3	6157.2	6208.7	6234.6	6260.6	6286.7	6302.5	6313.0	6334.1	6339.3
2	98.502	99.000	99.166	99.249	99.299	99.332	99.356	99.374	99.388	99.399	99.415	99.432	99.449	99.457	99.465	99.474	99.479	99.482	99.489	99.490
3	34.116	30.816	29.456	28.709	28.237	27.910	27.671	27.489	27.345	27.228	27.051	26.872	26.689	26.597	26.504	26.410	26.354	26.316	26.240	26.221
4	21.197	18.000	16.694	15.977	15.521	15.206	14.975	14.798	14.659	14.545	14.373	14.198	14.019	13.929	13.837	13.745	13.689	13.652	13.577	13.558
5	16.258	13.273	12.060	11.391	10.967	10.672	10.455	10.289	10.157	10.051	9.8883	9.7222	9.5526	9.4665	9.3793	9.2912	9.2378	9.2020	9.1299	9.1118
6	13.745	10.924	9.7795	9.1483	8.7459	8.4661	8.2600	8.1017	7.9761	7.8741	7.7183	7.5590	7.3958	7.3127	7.2285	7.1432	7.0915	7.0567	6.9867	6.9690
7	12.246	9.5466	8.4513	7.8466	7.4604	7.1914	6.9928	6.8400	6.7188	6.6201	6.4691	6.3143	6.1554	6.0743	5.9920	5.9084	5.8577	5.8236	5.7547	5.7373
8	11.258	8.6491	7.5910	7.0061	6.6318	6.3707	6.1776	6.0289	5.9106	5.8143	5.6667	5.5151	5.3591	5.2793	5.1981	5.1156	5.0654	5.0316	4.9633	4.9461
9	10.561	8.0215	6.9919	6.4221	6.0569	5.8018	5.6129	5.4671	5.3511	5.2565	5.1114	4.9621	4.8080	4.7290	4.6486	4.5666	4.5167	4.4831	4.4150	4.3978
10	10.044	7.5594	6.5523	5.9943	5.6363	5.3858	5.2001	5.0567	4.9424	4.8491	4.7059	4.5581	4.4054	4.3269	4.2469	4.1653	4.1155	4.0819	4.0137	3.9965
11	9.6460	7.2057	6.2167	5.6683	5.3160	5.0692	4.8861	4.7445	4.6315	4.5393	4.3974	4.2509	4.0990	4.0209	3.9411	3.8596	3.8097	3.7761	3.7077	3.6904
12	9.3302	6.9266	5.9525	5.4120	5.0643	4.8206	4.6395	4.4994	4.3875	4.2961	4.1553	4.0096	3.8584	3.7805	3.7008	3.6192	3.5692	3.5355	3.4668	3.4494
13	9.0738	6.7010	5.7394	5.2053	4.8616	4.6204	4.4410	4.3021	4.1911	4.1003	3.9603	3.8154	3.6646	3.5868	3.5070	3.4253	3.3752	3.3413	3.2723	3.2548
14	8.8616	6.5149	5.5639	5.0354	4.6950	4.4558	4.2779	4.1399	4.0297	3.9394	3.8001	3.6557	3.5052	3.4274	3.3476	3.2656	3.2153	3.1813	3.1118	3.0942
15	8.6831	6.3589	5.4170	4.8932	4.5556	4.3183	4.1415	4.0045	3.8948	3.8049	3.6662	3.5222	3.3719	3.2940	3.2141	3.1319	3.0814	3.0471	2.9772	2.9595
16	8.5310	6.2262	5.2922	4.7726	4.4374	4.2016	4.0259	3.8896	3.7804	3.6909	3.5527	3.4089	3.2587	3.1808	3.1007	3.0182	2.9675	2.9330	2.8627	2.8447
17	8.3997	6.1121	5.1850	4.6690	4.3359	4.1015	3.9267	3.7910	3.6822	3.5931	3.4552	3.3117	3.1615	3.0835	3.0032	2.9205	2.8694	2.8348	2.7639	2.7459
18	8.2854	6.0129	5.0919	4.5790	4.2479	4.0146	3.8406	3.7054	3.5971	3.5082	3.3706	3.2273	3.0771	2.9990	2.9185	2.8354	2.7841	2.7493	2.6779	2.6597
19	8.1849	5.9259	5.0103	4.5003	4.1708	3.9386	3.7653	3.6305	3.5225	3.4338	3.2965	3.1533	3.0031	2.9249	2.8442	2.7608	2.7093	2.6742	2.6023	2.5839
20	8.0960	5.8489	4.9382	4.4307	4.1027	3.8714	3.6987	3.5644	3.4567	3.3682	3.2311	3.0880	2.9377	2.8594	2.7785	2.6947	2.6430	2.6077	2.5353	2.5168
21	8.0166	5.7804	4.8740	4.3688	4.0421	3.8117	3.6396	3.5056	3.3981	3.3098	3.1730	3.0300	2.8796	2.8010	2.7200	2.6359	2.5838	2.5484	2.4755	2.4568
22	7.9454	5.7190	4.8166	4.3134	3.9880	3.7583	3.5867	3.4530	3.3458	3.2576	3.1209	2.9779	2.8274	2.7488	2.6675	2.5831	2.5308	2.4951	2.4217	2.4029
23	7.8811	5.6637	4.7649	4.2636	3.9392	3.7102	3.5390	3.4057	3.2986	3.2106	3.0740	2.9311	2.7805	2.7017	2.6202	2.5355	2.4829	2.4471	2.3732	2.3542

（续表）

m	n=1	2	3	4	5	6	7	8	9	10	12	15	20	24	30	40	50	60	100	120
24	7.8229	5.6136	4.7181	4.2184	3.8951	3.6667	3.4959	3.3629	3.2560	3.1681	3.0316	2.8887	2.7380	2.6591	2.5773	2.4923	2.4395	2.4035	2.3291	2.3100
25	7.7698	5.5680	4.6755	4.1774	3.8550	3.6272	3.4568	3.3239	3.2172	3.1294	2.9931	2.8502	2.6993	2.6203	2.5383	2.4530	2.3999	2.3637	2.2888	2.2696
26	7.7213	5.5263	4.6366	4.1400	3.8183	3.5911	3.4210	3.2884	3.1818	3.0941	2.9578	2.8150	2.6640	2.5848	2.5026	2.4170	2.3637	2.3273	2.2519	2.2325
27	7.6767	5.4881	4.6009	4.1056	3.7848	3.5580	3.3882	3.2558	3.1494	3.0618	2.9256	2.7827	2.6316	2.5522	2.4699	2.3840	2.3304	2.2938	2.2180	2.1985
28	7.6356	5.4529	4.5681	4.0740	3.7539	3.5276	3.3581	3.2259	3.1195	3.0320	2.8959	2.7530	2.6017	2.5223	2.4397	2.3535	2.2997	2.2629	2.1867	2.1670
29	7.5977	5.4204	4.5378	4.0449	3.7254	3.4995	3.3303	3.1982	3.0920	3.0045	2.8685	2.7256	2.5742	2.4946	2.4118	2.3253	2.2714	2.2344	2.1577	2.1379
30	7.5625	5.3903	4.5097	4.0179	3.6990	3.4735	3.3045	3.1726	3.0665	2.9791	2.8431	2.7002	2.5487	2.4689	2.3860	2.2992	2.2450	2.2079	2.1307	2.1108
40	7.3141	5.1785	4.3126	3.8283	3.5138	3.2910	3.1238	2.9930	2.8876	2.8005	2.6648	2.5216	2.3689	2.2880	2.2034	2.1142	2.0581	2.0194	1.9383	1.9172
50	7.1706	5.0566	4.1993	3.7195	3.4077	3.1864	3.0202	2.8900	2.7850	2.6981	2.5625	2.4190	2.2652	2.1835	2.0976	2.0066	1.9490	1.9090	1.8248	1.8026
60	7.0771	4.9774	4.1259	3.6490	3.3389	3.1187	2.9530	2.8233	2.7185	2.6318	2.4961	2.3523	2.1978	2.1154	2.0285	1.9360	1.8772	1.8363	1.7493	1.7263
70	7.0114	4.9219	4.0744	3.5996	3.2907	3.0712	2.9060	2.7765	2.6719	2.5852	2.4496	2.3055	2.1504	2.0674	1.9797	1.8861	1.8263	1.7846	1.6954	1.6717
80	6.9627	4.8807	4.0363	3.5631	3.2550	3.0361	2.8713	2.7420	2.6374	2.5508	2.4151	2.2709	2.1153	2.0318	1.9435	1.8489	1.7883	1.7459	1.6548	1.6305
90	6.9251	4.8491	4.0070	3.5350	3.2276	3.0091	2.8445	2.7154	2.6109	2.5243	2.3886	2.2442	2.0882	2.0044	1.9155	1.8201	1.7588	1.7158	1.6231	1.5982
100	6.8953	4.8239	3.9837	3.5127	3.2059	2.9877	2.8233	2.6943	2.5898	2.5033	2.3676	2.2230	2.0666	1.9826	1.8933	1.7972	1.7353	1.6918	1.5977	1.5723
110	6.8710	4.8035	3.9648	3.4946	3.1882	2.9703	2.8061	2.6771	2.5727	2.4862	2.3505	2.2058	2.0491	1.9648	1.8751	1.7784	1.7160	1.6721	1.5767	1.5509
120	6.8509	4.7865	3.9491	3.4795	3.1735	2.9559	2.7918	2.6629	2.5586	2.4721	2.3363	2.1915	2.0346	1.9500	1.8600	1.7628	1.7000	1.6557	1.5592	1.5330
140	6.8194	4.7600	3.9246	3.4561	3.1507	2.9333	2.7695	2.6407	2.5365	2.4500	2.3142	2.1692	2.0119	1.9269	1.8364	1.7384	1.6748	1.6299	1.5315	1.5046
160	6.7960	4.7403	3.9064	3.4386	3.1336	2.9166	2.7528	2.6242	2.5200	2.4335	2.2977	2.1526	1.9949	1.9097	1.8187	1.7201	1.6559	1.6105	1.5106	1.4832
180	6.7778	4.7250	3.8923	3.4251	3.1205	2.9036	2.7400	2.6114	2.5072	2.4208	2.2849	2.1397	1.9818	1.8963	1.8050	1.7059	1.6413	1.5954	1.4942	1.4663
200	6.7633	4.7129	3.8810	3.4143	3.1100	2.8933	2.7298	2.6012	2.4971	2.4106	2.2747	2.1294	1.9713	1.8857	1.7941	1.6945	1.6295	1.5833	1.4811	1.4527

参考文献

[1] 高德纳 . 计算机程序设计艺术（第 3 版）. 北京：人民邮电出版社，2016.

[2] 同济大学数学系编 . 工程数学：线性代数（第 6 版）. 北京：高等教育出版社，2014.

[3] 李航 . 统计学习方法 . 北京：清华大学出版社，2012.

[4] 盛骤，谢式千，潘承毅 . 概率论与数理统计（第 4 版）. 北京：高等教育出版社，2010.

[5] 何书元 . 概率论 . 北京：北京大学出版社，2006.

[6] 陈家鼎 . 数理统计学讲义（第 2 版）. 北京：高等教育出版社，2006.

[7] 茆诗松，程依明 . 濮晓龙，等 . 概率论与数理统计教程（第 2 版）. 北京：高等教育出版社，2004.

[8] 程士宏，等 . 测度论和概率论基础 . 北京：北京大学出版社，2004.

[9] [美] C.R. 劳 . 统计与真理：怎样运用偶然性 . 北京：科学出版社，2004.

[10] 陈希孺 . 数理统计学简史 . 长沙：湖南教育出版社，2002.

[11] 徐振邦，娄元仁，等 . 数学地质基础 . 北京：北京大学出版社，1994.

[12] 李成瑞 . 社会经济统计学原理教程 . 北京：中国统计出版社，1992.

[13] 陈善林，张浙，等 . 统计发展史 . 上海：立信会计图书用品社，1987.

[14] 杨坚白，等 . 统计学原理 . 上海：上海人民出版社，1987.

[15] 高庆丰 . 欧美统计学史 . 北京：中国统计出版社，1987.

[16] Eric Lehman, F Thomson Leighton, Albert R Meyer. Mathematics for Computer Science. Google and MIT, 2017.

[17] LoraD.DelWiche, Susan J. Slaughter The Little SAS Book: A primer, FIFTH EDITION: SAS Institute, 2011.

[18] Andrew J. Vickers Whatis a p-value anyway? 34 Stories to Help You Actually Understand Statistics.2009.

[19] Catherine Truxillo, Robert Hamer Multivariate Statistical Methods: Practical Research Applications Course Notes. SAS Institute Inc, 2007.

[20] Andrew Gelman, Jennifer Hill. Data Analysis Using Regression and Multilevel/Hierarchical Models. 2006.

[21] David Freedman, Robert Pisani, Roger Purves.Statistics (4th Edition). 1978.

[22] Jonathan M. Borwein and Peter B. Borwein, Pi and the AGM — A study in Analytic Number Theory and Computational Complexity, A Wiley-Interscience Publication. New York, 1987.

[23] LAY-YONG Lam and TIAN-SE Ang. Circle Measurements in Ancient China. Historia Mathematica, 1986.

[24] Engels, Hermann. Quadrature of the Circle in Ancient Egypt. Historia Mathematica, 1977.

[25] William Feller.An Introduction to Probability Theory and Its Applications.New York: Wiley, 1971.